J. H. Ferziger
M. Perić

Computational Methods for Fluid Dynamics

3rd Edition

Joel H. Ferziger / Milovan Perić

Computational Methods for Fluid Dynamics

Joel H. Ferziger / Milovan Perić

Computational Methods for Fluid Dynamics

third, rev. edition

With 128 Figures

Springer

Preface

Computational fluid dynamics, commonly known by the acronym 'CFD', is undergoing significant expansion in terms of both the number of courses offered at universities and the number of researchers active in the field. There are a number of software packages available that solve fluid flow problems; the market is not quite as large as the one for structural mechanics codes, in which finite element methods are well established. The lag can be explained by the fact that CFD problems are, in general, more difficult to solve. However, CFD codes are slowly being accepted as design tools by industrial users. At present, users of CFD need to be fairly knowledgeable, which requires education of both students and working engineers. The present book is an attempt to fill this need.

It is our belief that, to work in CFD, one needs a solid background in both fluid mechanics and numerical analysis; significant errors have been made by people lacking knowledge in one or the other. We therefore encourage the reader to obtain a working knowledge of these subjects before entering into a study of the material in this book. Because different people view numerical methods differently, and to make this work more self-contained, we have included two chapters on basic numerical methods in this book. The book is based on material offered by the authors in courses at Stanford University, the University of Erlangen-Nürnberg and the Technical University of Hamburg-Harburg. It reflects the authors' experience in both writing CFD codes and using them to solve engineering problems. Many of the codes used in the examples, from the simple ones involving rectangular grids to the ones using non-orthogonal grids and multigrid methods, are available to interested readers; see the information on how to access them via Internet in the appendix. These codes illustrate the methods described in the book; they can be adapted to the solution of many fluid mechanical problems. Students should try to modify them (e.g. to implement different boundary conditions, interpolation schemes, differentiation and integration approximations, etc.). This is important as one does not really know a method until s/he has programmed and/or run it.

Since one of the authors (M.P.) has just recently decided to give up his professor position to work for a provider of CFD tools, we have also included in the Internet site a special version of a full-featured commercial CFD package

that can be used to solve many different flow problems. This is accompanied by a collection of prepared and solved test cases that are suitable to learn how to use such tools most effectively. Experience with this tool will be valuable to anyone who has never used such tools before, as the major issues are common to most of them. Suggestions are also given for parameter variation, error estimation, grid quality assessment, and efficiency improvement.

The finite volume method is favored in this book, although finite difference methods are described in what we hope is sufficient detail. Finite element methods are not covered in detail as a number of books on that subject already exist.

We have tried to describe the basic ideas of each topic in such a way that they can be understood by the reader; where possible, we have avoided lengthy mathematical analysis. Usually a general description of an idea or method is followed by a more detailed description (including the necessary equations) of one or two numerical schemes representative of the better methods of the type; other possible approaches and extensions are briefly described. We have tried to emphasize common elements of methods rather than their differences.

There is a vast literature devoted to numerical methods for fluid mechanics. Even if we restrict our attention to incompressible flows, it would be impossible to cover everything in a single work. Doing so would create confusion for the reader. We have therefore covered only the methods that we have found valuable and that are commonly used in industry in this book. References to other methods are given, however.

We have placed considerable emphasis on the need to estimate numerical errors; almost all examples in this book are accompanied with error analysis. Although it is possible for a qualitatively incorrect solution of a problem to look reasonable (it may even be a good solution of another problem), the consequences of accepting it may be severe. On the other hand, sometimes a relatively poor solution can be of value if treated with care. Industrial users of commercial codes need to learn to judge the quality of the results before believing them; we hope that this book will contribute to the awareness that numerical solutions are always approximate.

We have tried to cover a cross-section of modern approaches, including direct and large eddy simulation of turbulence, multigrid methods and parallel computing, methods for moving grids and free surface flows, etc. Obviously, we could not cover all these topics in detail, but we hope that the information contained herein will provide the reader with a general knowledge of the subject; those interested in a more detailed study of a particular topic will find recommendations for further reading.

While we have invested every effort to avoid typing, spelling and other errors, no doubt some remain to be found by readers. We will appreciate your notifying us of any mistakes you might find, as well as your comments and suggestions for improvement of future editions of the book. For that

purpose, the authors' electronic mail addresses are given below. We also hope that colleagues whose work has not been referenced will forgive us, since any omissions are unintentional.

We have to thank all our present and former students, colleagues, and friends, who helped us in one way or another to finish this work; the complete list of names is too long to list here. Names that we cannot avoid mentioning include Drs. Ismet Demirdžić, Samir Muzaferija, Željko Lilek, Joseph Oliger, Gene Golub, Eberhard Schreck, Volker Seidl, Kishan Shah, Fotina (Tina) Katapodes and David Briggs. The help provided by those people who created and made available TeX, LaTeX, Linux, Xfig, Ghostscript and other tools which made our job easier is also greatly appreciated.

Our families gave us a tremendous support during this endeavor; our special thanks go to Anna, Robinson and Kerstin Perić and Eva Ferziger.

This collaboration between two geographically distant colleagues was made possible by grants and fellowships from the Alexander von Humboldt Foundation and the Deutsche Forschungsgemeinschaft (German National Research Organization). Without their support, this work would never have come into existence and we cannot express sufficient thanks to them.

Milovan Perić
milovan@cd.co.uk
Joel H. Ferziger
ferziger@leland.stanford.edu

Contents

1. Basic Concepts of Fluid Flow

1.1 Introduction

Fluids are substances whose molecular structure offers no resistance to external shear forces: even the smallest force causes *deformation* of a fluid particle. Although a significant distinction exists between *liquids* and *gases*, both types of fluids obey the same laws of motion. In most cases of interest, a fluid can be regarded as *continuum*, i.e. a continuous substance.

Fluid flow is caused by the action of externally applied forces. Common driving forces include pressure differences, gravity, shear, rotation, and surface tension. They can be classified as *surface forces* (e.g. the shear force due to wind blowing above the ocean or pressure and shear forces created by a movement of a rigid wall relative to the fluid) and *body forces* (e.g. gravity and forces induced by rotation).

While all fluids behave similarly under action of forces, their *macroscopic properties* differ considerably. These properties must be known if one is to study fluid motion; the most important properties of simple fluids are the *density* and *viscosity*. Others, such as *Prandtl number, specific heat*, and *surface tension* affect fluid flows only under certain conditions, e.g. when there are large temperature differences. Fluid properties are functions of other thermodynamic variables (e.g. temperature and pressure); although it is possible to estimate some of them from statistical mechanics or kinetic theory, they are usually obtained by laboratory measurement.

Fluid mechanics is a very broad field. A small library of books would be required to cover all of the topics that could be included in it. In this book we shall be interested mainly in flows of interest to mechanical engineers but even that is a very broad area so we shall try to classify the types of problems that may be encountered. A more mathematical, but less complete, version of this scheme will be found in Sect. 1.8.

The speed of a flow affects its properties in a number of ways. At low enough speeds, the inertia of the fluid may be ignored and we have *creeping flow*. This regime is of importance in flows containing small particles (suspensions), in flows through porous media or in narrow passages (coating techniques, micro-devices). As the speed is increased, inertia becomes important but each fluid particle follows a smooth trajectory; the flow is then said to be *laminar*. Further increases in speed may lead to instability that

eventually produces a more random type of flow that is called *turbulent*; the process of laminar-turbulent *transition* is an important area in its own right. Finally, the ratio of the flow speed to the speed of sound in the fluid (the *Mach number*) determines whether exchange between kinetic energy of the motion and internal degrees of freedom needs to be considered. For small Mach numbers, Ma < 0.3, the flow may be considered *incompressible*; otherwise, it is *compressible*. If Ma < 1, the flow is called *subsonic*; when Ma > 1, the flow is *supersonic* and shock waves are possible. Finally, for Ma > 5, the compression may create high enough temperatures to change the chemical nature of the fluid; such flows are called *hypersonic*. These distinctions affect the mathematical nature of the problem and therefore the solution method. Note that we call the flow compressible or incompressible depending on the Mach number, even though compressibility is a property of the fluid. This is common terminology since the flow of a compressible fluid at low Mach number is essentially incompressible.

In many flows, the effects of viscosity are important only near walls, so that the flow in the largest part of the domain can be considered as *inviscid*. In the fluids we treat in this book, Newton's law of viscosity is a good approximation and it will be used exclusively. Fluids obeying Newton's law are called *Newtonian*; *non-Newtonian* fluids are important for some engineering applications but are not treated here.

Many other phenomena affect fluid flow. These include temperature differences which lead to *heat transfer* and density differences which give rise to *buoyancy*. They, and differences in concentration of solutes, may affect flows significantly or, even be the sole cause of the flow. Phase changes (boiling, condensation, melting and freezing), when they occur, always lead to important modifications of the flow and give rise to *multi-phase* flow. Variation of other properties such as viscosity, surface tension etc. may also play important role in determining the nature of the flow. With only a few exceptions, these effects will not be considered in this book.

In this chapter the basic equations governing fluid flow and associated phenomena will be presented in several forms: *(i)* a coordinate-free form, which can be specialized to various coordinate systems, *(ii)* an integral form for a finite control volume, which serves as starting point for an important class of numerical methods, and *(iii)* a differential (tensor) form in a Cartesian reference frame, which is the basis for another important approach. The basic conservation principles and laws used to derive these equations will only be briefly summarized here; more detailed derivations can be found in a number of standard texts on fluid mechanics (e.g. Bird et al., 1962; Slattery, 1972; White, 1986). It is assumed that the reader is somewhat familiar with the physics of fluid flow and related phenomena, so we shall concentrate on techniques for the numerical solution of the governing equations.

1.2 Conservation Principles

Conservation laws can be derived by considering a given quantity of matter or *control mass* (CM) and its *extensive* properties, such as mass, momentum and energy. This approach is used to study the dynamics of solid bodies, where the CM (sometimes called the *system*) is easily identified. In fluid flows, however, it is difficult to follow a parcel of matter. It is more convenient to deal with the flow within a certain spatial region we call a *control volume* (CV), rather than in a parcel of matter which quickly passes through the region of interest. This method of analysis is called the *control volume approach.*

We shall be concerned primarily with two extensive properties, mass and momentum. The conservation equations for these and other properties have common terms which will be considered first.

The conservation law for an extensive property relates the rate of change of the amount of that property in a given control mass to externally determined effects. For mass, which is neither created nor destroyed in the flows of engineering interest, the conservation equation can be written:

$$\frac{\mathrm{d}m}{\mathrm{d}t} = 0 \, . \tag{1.1}$$

On the other hand, momentum can be changed by the action of forces and its conservation equation is Newton's second law of motion:

$$\frac{\mathrm{d}(m\boldsymbol{v})}{\mathrm{d}t} = \sum \boldsymbol{f} \, , \tag{1.2}$$

where t stands for time, m for mass, $\boldsymbol{v}$ for the velocity, and $\boldsymbol{f}$ for forces acting on the control mass.

We shall transform these laws into a control volume form that will be used throughout this book. The fundamental variables will be *intensive* rather than extensive properties; the former are properties which are independent of the amount of matter considered. Examples are density ρ (mass per unit volume) and velocity $\boldsymbol{v}$ (momentum per unit mass).

If ϕ is any conserved intensive property (for mass conservation, $\phi = 1$; for momentum conservation, $\phi = \boldsymbol{v}$; for conservation of a scalar, ϕ represents the conserved property per unit mass), then the corresponding extensive property Φ can be expressed as:

$$\Phi = \int_{\Omega_{\mathrm{CM}}} \rho\phi \, \mathrm{d}\Omega \, , \tag{1.3}$$

where Ω_{CM} stands for volume occupied by the CM. Using this definition, the left hand side of each conservation equation for a control volume can be written:[1]

[1] This equation is often called *control volume equation* or the *Reynolds' transport theorem.*

$$\frac{\mathrm{d}}{\mathrm{d}t} \int\limits_{\Omega_{\mathrm{CM}}} \rho\phi \,\mathrm{d}\Omega = \frac{\mathrm{d}}{\mathrm{d}t} \int\limits_{\Omega_{\mathrm{CV}}} \rho\phi \,\mathrm{d}\Omega + \int\limits_{S_{\mathrm{CV}}} \rho\phi \,(\boldsymbol{v} - \boldsymbol{v}_{\mathrm{b}}) \cdot \boldsymbol{n} \,\mathrm{d}S \;, \tag{1.4}$$

where Ω_{CV} is the CV volume, S_{CV} is the surface enclosing CV, $\boldsymbol{n}$ is the unit vector orthogonal to S_{CV} and directed outwards, $\boldsymbol{v}$ is the fluid velocity and $\boldsymbol{v}_{\mathrm{b}}$ is the velocity with which the CV surface is moving. For a fixed CV, which we shall be considering most of the time, $\boldsymbol{v}_{\mathrm{b}} = \boldsymbol{0}$ and the first derivative on the right hand side becomes a local (partial) derivative. This equation states that the rate of change of the amount of the property in the control mass, Φ, is the rate of change of the property within the control volume plus the net flux of it through the CV boundary due to fluid motion relative to CV boundary. The last term is usually called the *convective* (or sometimes, advective) flux of ϕ through the CV boundary. If the CV moves so that its boundary coincides with the boundary of a control mass, then $\boldsymbol{v} = \boldsymbol{v}_{\mathrm{b}}$ and this term will be zero as required.

A detailed derivation of this equation is given in in many textbooks on fluid dynamics (e.g. in Bird et al., 1962; Fox and McDonald, 1982) and will not be repeated here. The mass, momentum and scalar conservation equations will be presented in the next three sections. For convenience, a fixed CV will be considered; Ω represents the CV volume and S its surface.

1.3 Mass Conservation

The integral form of the mass conservation (continuity) equation follows directly from the control volume equation, by setting $\phi = 1$:

$$\frac{\partial}{\partial t} \int_{\Omega} \rho \,\mathrm{d}\Omega + \int_{S} \rho \boldsymbol{v} \cdot \boldsymbol{n} \,\mathrm{d}S = 0 \;. \tag{1.5}$$

By applying the Gauss' divergence theorem to the convection term, we can transform the surface integral into a volume integral. Allowing the control volume to become infinitesimally small leads to a differential coordinate-free form of the continuity equation:

$$\frac{\partial \rho}{\partial t} + \operatorname{div}(\rho \boldsymbol{v}) = 0 \;. \tag{1.6}$$

This form can be transformed into a form specific to a given coordinate system by providing the expression for the divergence operator in that system. Expressions for common coordinate systems such as the Cartesian, cylindrical and spherical systems can be found in many textbooks (e.g. Bird et al., 1962); expressions applicable to general non-orthogonal coordinate systems are given e.g. in Truesdell (1977), Aris (1989), Sedov (1971). We present below the Cartesian form in both tensor and expanded notation. Here and throughout this book we shall adopt the Einstein convention that whenever the same

index appears twice in any term, summation over the range of that index is implied:

$$\frac{\partial \rho}{\partial t} + \frac{\partial (\rho u_i)}{\partial x_i} = \frac{\partial \rho}{\partial t} + \frac{\partial (\rho u_x)}{\partial x} + \frac{\partial (\rho u_y)}{\partial y} + \frac{\partial (\rho u_z)}{\partial z} = 0 \,, \tag{1.7}$$

where x_i (i=1,2,3) or (x, y, z) are the Cartesian coordinates and u_i or (u_x, u_y, u_z) are the Cartesian components of the velocity vector $\boldsymbol{v}$. The conservation equations in Cartesian form are often used and this will be the case in this work. Differential conservation equations in non-orthogonal coordinates will be presented in Chap. 8.

1.4 Momentum Conservation

There are several ways of deriving the momentum conservation equation. One approach is to use the control volume method described in Sect. 1.2; in this method, one uses Eqs. (1.2) and (1.4) and replaces ϕ by $\boldsymbol{v}$, e.g. for a fixed fluid-containing volume of space:

$$\frac{\partial}{\partial t} \int_{\Omega} \rho \boldsymbol{v} \, \mathrm{d}\Omega + \int_{S} \rho \boldsymbol{v} \boldsymbol{v} \cdot \boldsymbol{n} \, \mathrm{d}S = \sum \boldsymbol{f} \,. \tag{1.8}$$

To express the right hand side in terms of intensive properties, one has to consider the forces which may act on the fluid in a CV:

- surface forces (pressure, normal and shear stresses, surface tension etc.);
- body forces (gravity, centrifugal and Coriolis forces, electromagnetic forces, etc.).

The surface forces due to pressure and stresses are, from the molecular point of view, the microscopic momentum fluxes across a surface. If these fluxes cannot be written in terms of the properties whose conservation the equations govern (density and velocity), the system of equations is not closed; that is there are fewer equations than dependent variables and solution is not possible. This possibility can be avoided by making certain assumptions. The simplest assumption is that the fluid is Newtonian; fortunately, the Newtonian model applies to many actual fluids.

For Newtonian fluids, the stress tensor T, which is the molecular rate of transport of momentum, can be written:

$$\mathsf{T} = -\left(p + \frac{2}{3}\mu \, \mathrm{div} \, \boldsymbol{v}\right) \mathsf{I} + 2\mu \mathsf{D} \,, \tag{1.9}$$

where μ is the dynamic viscosity, I is the unit tensor, p is the static pressure and D is the rate of strain (deformation) tensor:

$$\mathsf{D} = \frac{1}{2}\left[\operatorname{grad}\boldsymbol{v} + (\operatorname{grad}\boldsymbol{v})^T\right] . \tag{1.10}$$

These two equations may be written, in index notation in Cartesian coordinates, as follows:

$$T_{ij} = -\left(p + \frac{2}{3}\mu\frac{\partial u_j}{\partial x_j}\right)\delta_{ij} + 2\mu D_{ij} , \tag{1.11}$$

$$D_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) , \tag{1.12}$$

where δ_{ij} is Kronecker symbol ($\delta_{ij} = 1$ if $i = j$ and $\delta_{ij} = 0$ otherwise). For incompressible flows, the second term in brackets in Eq. (1.11) is zero by virtue of the continuity equation. The following notation is often used in literature to describe the viscous part of the stress tensor:

$$\tau_{ij} = 2\mu D_{ij} - \frac{2}{3}\mu\delta_{ij}\operatorname{div}\boldsymbol{v} . \tag{1.13}$$

For non-Newtonian fluids, the relation between the stress tensor and the velocity is defined by a set of partial differential equations and the total problem is far more complicated. In fact, different types of non-Newtonian fluids require different constitutive equations which may, in turn, require different solution methods. This subject is complex and is just beginning to be explored. For these reasons, it will not be considered further in this book.

With the body forces (per unit mass) being represented by $\boldsymbol{b}$, the integral form of the momentum conservation equation becomes:

$$\frac{\partial}{\partial t}\int_\Omega \rho\boldsymbol{v}\,\mathrm{d}\Omega + \int_S \rho\boldsymbol{v}\boldsymbol{v}\cdot\boldsymbol{n}\,\mathrm{d}S = \int_S \mathsf{T}\cdot\boldsymbol{n}\,\mathrm{d}S + \int_\Omega \rho\boldsymbol{b}\,\mathrm{d}\Omega . \tag{1.14}$$

A coordinate-free vector form of the momentum conservation equation (1.14) is readily obtained by applying Gauss' divergence theorem to the convective and diffusive flux terms:

$$\frac{\partial(\rho\boldsymbol{v})}{\partial t} + \operatorname{div}(\rho\boldsymbol{v}\boldsymbol{v}) = \operatorname{div}\mathsf{T} + \rho\boldsymbol{b} . \tag{1.15}$$

The corresponding equation for the ith Cartesian component is:

$$\frac{\partial(\rho u_i)}{\partial t} + \operatorname{div}(\rho u_i\boldsymbol{v}) = \operatorname{div}\boldsymbol{t}_i + \rho b_i . \tag{1.16}$$

Since momentum is a vector quantity, the convective and diffusive fluxes of it through a CV boundary are the scalar products of second rank tensors ($\rho\boldsymbol{v}\boldsymbol{v}$ and T) with the surface vector $\boldsymbol{n}\,\mathrm{d}S$. The integral form of the above equations is:

$$\frac{\partial}{\partial t}\int_{\Omega} \rho u_i \, d\Omega + \int_S \rho u_i \boldsymbol{v} \cdot \boldsymbol{n} \, dS = \int_S \boldsymbol{t}_i \cdot \boldsymbol{n} \, dS + \int_{\Omega} \rho b_i \, d\Omega \,, \tag{1.17}$$

where (see Eqs. (1.9) and (1.10)):

$$\boldsymbol{t}_i = \mu \operatorname{grad} u_i + \mu (\operatorname{grad} \boldsymbol{v})^T \cdot \boldsymbol{i}_i - \left(p + \frac{2}{3}\mu \operatorname{div} \boldsymbol{v}\right) \boldsymbol{i}_i = \tau_{ij}\boldsymbol{i}_j - p\boldsymbol{i}_i \,. \tag{1.18}$$

Here b_i stands for the ith component of the body force, superscript ^{T} means transpose and $\boldsymbol{i}_i$ is the Cartesian unit vector in the direction of the coordinate x_i. In Cartesian coordinates one can write the above expression as:

$$\boldsymbol{t}_i = \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \boldsymbol{i}_j - \left(p + \frac{2}{3}\mu \frac{\partial u_j}{\partial x_j}\right) \boldsymbol{i}_i \,. \tag{1.19}$$

A vector field can be represented in a number of different ways. The basis vectors in terms of which the vector is defined may be local or global. In curvilinear coordinate systems, which are often required when the boundaries are complex (see Chap. 8) one may choose either a covariant or a contravariant basis, see Fig. 1.1. The former expresses a vector in terms of its components along the local coordinates; the latter uses the projections normal to coordinate surfaces. In a Cartesian system, the two become identical. Also, the basis vectors may be dimensionless or dimensional. Including all of these options, over 70 different forms of the momentum equations are possible. Mathematically, all are equivalent; from the numerical point of view, some are more difficult to deal with than others.
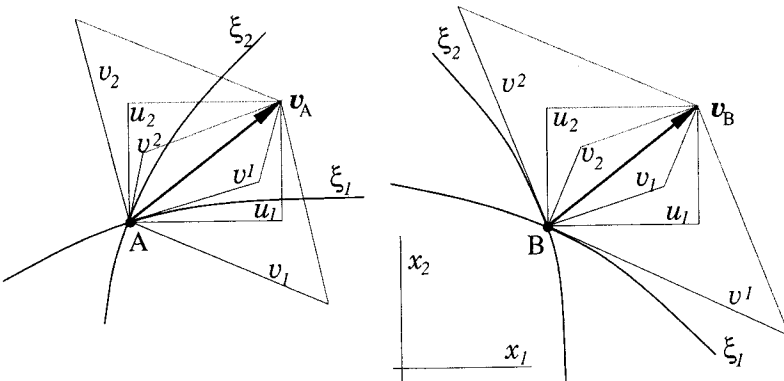
Fig. 1.1. Representation of a vector through different components: u_i – Cartesian components; v^i – contravariant components; v_i – covariant components [$\boldsymbol{v}_A = \boldsymbol{v}_B$, $(u_i)_A = (u_i)_B$, $(v^i)_A \neq (v^i)_B$, $(v_i)_A \neq (v_i)_B$]

The momentum equations are said to be in "strong conservation form" if all terms have the form of the divergence of a vector or tensor. This is possi-

ble for the component form of the equations only when components in fixed directions are used. A coordinate-oriented vector component turns with the coordinate direction and an "apparent force" is required to produce the turning; these forces are non-conservative in the sense defined above. For example, in cylindrical coordinates the radial and circumferential directions change so the components of a spatially constant vector (e.g. a uniform velocity field) vary with r and θ and are singular at the coordinate origin. To account for this, the equations in terms of these components contain centrifugal and Coriolis force terms.

Figure 1.1 shows a vector $\boldsymbol{v}$ and its contravariant, covariant and Cartesian components. Obviously, the contravariant and covariant components change as the base vectors change even though the vector $\boldsymbol{v}$ remains constant. We shall discuss the effect of the choice of velocity components on numerical solution methods in Chap. 8.

The strong conservation form of the equations, when used together with a finite volume method, automatically insures global momentum conservation in the calculation. This is an important property of the conservation equations and its preservation in the numerical solution is equally important. Retention of this property can help to insure that the numerical method will not diverge during the solution and may be regarded as a kind of "realizability".

For some flows it is advantageous to resolve the momentum in spatially variable directions. For example, the velocity in a line vortex has only one component u_θ in cylindrical coordinates but two components in Cartesian coordinates. Axisymmetric flow without swirl is two-dimensional (2D) when analyzed in a polar-cylindrical coordinate frame, but three-dimensional (3D) when a Cartesian frame is used. Some numerical techniques that use non-orthogonal coordinates require use of contravariant velocity components. The equations then contain so-called "curvature terms", which are hard to compute accurately because they contain second derivatives of the coordinate transformations that are difficult to approximate.

Throughout this book we shall work with velocity vectors and stress tensors in terms of their Cartesian components, and we shall use conservative form of the Cartesian momentum equations.

Equation (1.16) is in strong conservation form. A non-conservative form of this equation can be obtained by employing the continuity equation; since

$$\operatorname{div}(\rho \boldsymbol{v} u_i) = u_i \operatorname{div}(\rho \boldsymbol{v}) + \rho \boldsymbol{v} \cdot \operatorname{grad} u_i \ ,$$

it follows that:

$$\rho \frac{\partial u_i}{\partial t} + \rho \boldsymbol{v} \cdot \operatorname{grad} u_i = \operatorname{div} \boldsymbol{t}_i + \rho b_i \ . \tag{1.20}$$

The pressure term contained in $\boldsymbol{t}_i$ can also be written as

$$\operatorname{div}(p\, \boldsymbol{i}_i) = \operatorname{grad} p \cdot \boldsymbol{i}_i \ .$$

The pressure gradient is then regarded as a body force; this amounts to non-conservative treatment of the pressure term. The non-conservative form of equations is often used in finite difference methods, since it is somewhat simpler. In the limit of a very fine grid, all equation forms and numerical solution methods give the same solution; however, on coarse grids the non-conservative form introduces additional errors which may become important.

If the expression for the viscous part of the stress tensor, Eq. (1.13), is substituted into Eq. (1.16) written in index notation and for Cartesian coordinates, and if gravity is the only body force, one has:

$$\frac{\partial(\rho u_i)}{\partial t} + \frac{\partial(\rho u_j u_i)}{\partial x_j} = \frac{\partial \tau_{ij}}{\partial x_j} - \frac{\partial p}{\partial x_i} + \rho g_i \, , \tag{1.21}$$

where g_i is the component of the gravitational acceleration $\boldsymbol{g}$ in the direction of the Cartesian coordinate x_i. For the case of constant density and gravity, the term $\rho \boldsymbol{g}$ can be written as $\text{grad}\,(\rho \boldsymbol{g} \cdot \boldsymbol{r})$, where $\boldsymbol{r}$ is the position vector, $\boldsymbol{r} = x_i \boldsymbol{i}_i$ (usually, gravity is assumed to act in the negative z-direction, i.e. $\boldsymbol{g} = g_z \boldsymbol{k}$, g_z being negative; in this case $\boldsymbol{g} \cdot \boldsymbol{r} = g_z z$). Then $-\rho g_z z$ is the hydrostatic pressure, and it is convenient – and for numerical solution more efficient – to define $\tilde{p} = p - \rho g_z z$ as the *head* and use it in place of the pressure. The term ρg_i then disappears from the above equation. If the actual pressure is needed, one has only to add $\rho g_z z$ to $\tilde{p}$.

Since only the gradient of the pressure appears in the equation, the absolute value of the pressure is not important except in compressible flows.

In variable density flows (the variation of gravity can be neglected in all flows considered in this book), one can split the ρg_i term into two parts: $\rho_0 g_i + (\rho - \rho_0) g_i$, where ρ_0 is a reference density. The first part can then be included with pressure and if the density variation is retained only in the gravitational term, we have the *Boussinesq* approximation, see Sect. 1.7.

1.5 Conservation of Scalar Quantities

The integral form of the equation describing conservation of a scalar quantity, ϕ, is analogous to the previous equations and reads:

$$\frac{\partial}{\partial t} \int_{\Omega} \rho \phi \, \mathrm{d}\Omega + \int_S \rho \phi \boldsymbol{v} \cdot \boldsymbol{n} \, \mathrm{d}S = \sum f_\phi \, , \tag{1.22}$$

where f_ϕ represents transport of ϕ by mechanisms other than convection and any sources or sinks of the scalar. Diffusive transport is always present (even in stagnant fluids), and it is usually described by a gradient approximation, e.g. *Fourier's law* for heat diffusion and *Fick's law* for mass diffusion:

$$f_\phi^{\mathrm{d}} = \int_S \Gamma \, \text{grad}\, \phi \cdot \boldsymbol{n} \, \mathrm{d}S \, , \tag{1.23}$$

where Γ is the diffusivity for the quantity ϕ. An example is the energy equation which, for most engineering flows, can be written:

$$\frac{\partial}{\partial t}\int_{\Omega} \rho h \,\mathrm{d}\Omega + \int_S \rho h \boldsymbol{v}\cdot\boldsymbol{n}\,\mathrm{d}S = \int_S k \,\mathrm{grad}\, T\cdot\boldsymbol{n}\,\mathrm{d}S + \int_{\Omega}(\boldsymbol{v}\cdot\mathrm{grad}\,p + \mathsf{S} : \mathrm{grad}\,\boldsymbol{v})\,\mathrm{d}\Omega + \frac{\partial}{\partial t}\int_{\Omega} p\,\mathrm{d}\Omega\,, \tag{1.24}$$

where h is the enthalpy, T is the temperature, k is the thermal conductivity, $k = \mu c_\mathrm{p}/\mathrm{Pr}$, and S is the viscous part of the stress tensor, $\mathsf{S} = \mathsf{T} + p\mathsf{I}$. Pr is the Prandtl number and c_p is the specific heat at constant pressure. The source term represents work done by pressure and viscous forces; it may be neglected in incompressible flows. Further simplification is achieved by considering a fluid with constant specific heat, in which case a convection/diffusion equation for the temperature results:

$$\frac{\partial}{\partial t}\int_{\Omega} \rho T \,\mathrm{d}\Omega + \int_S \rho T \boldsymbol{v}\cdot\boldsymbol{n}\,\mathrm{d}S = \int_S \frac{\mu}{\mathrm{Pr}}\,\mathrm{grad}\, T\cdot\boldsymbol{n}\,\mathrm{d}S\,. \tag{1.25}$$

Species concentration equations have the same form, with T replaced by the concentration c and Pr replaced by Sc, the Schmidt number.

It is useful to write the conservation equations in a general form, as all of the above equations have common terms. The discretization and analysis can then be carried out in a general manner; when necessary, terms peculiar to an equation can be handled separately.

The integral form of the generic conservation equation follows directly from Eqs. (1.22) and (1.23):

$$\frac{\partial}{\partial t}\int_{\Omega} \rho\phi \,\mathrm{d}\Omega + \int_S \rho\phi \boldsymbol{v}\cdot\boldsymbol{n}\,\mathrm{d}S = \int_S \Gamma\,\mathrm{grad}\,\phi\cdot\boldsymbol{n}\,\mathrm{d}S + \int_{\Omega} q_\phi\,\mathrm{d}\Omega\,, \tag{1.26}$$

where q_ϕ is the source or sink of ϕ. The coordinate-free vector form of this equation is:

$$\frac{\partial(\rho\phi)}{\partial t} + \mathrm{div}\,(\rho\phi\boldsymbol{v}) = \mathrm{div}\,(\Gamma\,\mathrm{grad}\,\phi) + q_\phi\,. \tag{1.27}$$

In Cartesian coordinates and tensor notation, the differential form of the generic conservation equation is:

$$\frac{\partial(\rho\phi)}{\partial t} + \frac{\partial(\rho u_j\phi)}{\partial x_j} = \frac{\partial}{\partial x_j}\left(\Gamma\frac{\partial\phi}{\partial x_j}\right) + q_\phi\,. \tag{1.28}$$

Numerical methods will first be described for this generic conservation equation. Special features of the continuity and momentum equations (which are usually called *Navier-Stokes equations*) will be described afterwards as an extension of the methods for the generic equation.

1.6 Dimensionless Form of Equations

Experimental studies of flows are often carried out on models, and the results are displayed in dimensionless form, thus allowing scaling to real flow conditions. The same approach can be undertaken in numerical studies as well. The governing equations can be transformed to dimensionless form by using appropriate normalization. For example, velocities can be normalized by a reference velocity v_0, spatial coordinates by a reference length L_0, time by some reference time t_0, pressure by ρv_0^2, and temperature by some reference temperature difference $T_1 - T_0$. The dimensionless variables are then:

$$t^* = \frac{t}{t_0} \; ; \quad x_i^* = \frac{x_i}{L_0} \; ; \quad u_i^* = \frac{u_i}{v_0} \; ; \quad p^* = \frac{p}{\rho v_0^2} \; ; \quad T^* = \frac{T - T_0}{T_1 - T_0}$$

If the fluid properties are constant, the continuity, momentum and temperature equations are, in dimensionless form:

$$\frac{\partial u_i^*}{\partial x_i^*} = 0 \, , \tag{1.29}$$

$$\mathrm{St} \frac{\partial u_i^*}{\partial t^*} + \frac{\partial (u_j^* u_i^*)}{\partial x_j^*} = \frac{1}{\mathrm{Re}} \frac{\partial^2 u_i^*}{\partial x_j^{*2}} - \frac{\partial p^*}{\partial x_i^*} + \frac{1}{\mathrm{Fr}^2} \gamma_i \, , \tag{1.30}$$

$$\mathrm{St} \frac{\partial T^*}{\partial t^*} + \frac{\partial (u_j^* T^*)}{\partial x_j^*} = \frac{1}{\mathrm{Re \, Pr}} \frac{\partial^2 T^*}{\partial x_j^{*2}} \, . \tag{1.31}$$

The following dimensionless numbers appear in the equations:

$$\mathrm{St} = \frac{L_0}{v_0 t_0} \; ; \qquad \mathrm{Re} = \frac{\rho v_0 L_0}{\mu} \; ; \qquad \mathrm{Fr} = \frac{v_0}{\sqrt{L_0 g}} \, ,$$

which are called Strouhal, Reynolds, and Froude numbers, respectively. γ_i is the component of the normalized gravitational acceleration vector in the x_i direction.

For natural convection flows, the Boussinesq approximation is often used, in which case the last term in the momentum equations becomes:

$$\frac{\mathrm{Ra}}{\mathrm{Re}^2 \, \mathrm{Pr}} T^* \gamma_i \, ,$$

where Ra is the Rayleigh number, defined as:

$$\mathrm{Ra} = \frac{\rho^2 g \beta (T_1 - T_0) L_0^3}{\mu^2} \mathrm{Pr} \, ,$$

and β is the coefficient of thermal expansion.

The choice of the normalization quantities is obvious in simple flows; v_0 is the mean velocity and L_0 is a geometric length scale; T_0 and T_1 are the cold and hot wall temperatures. If the geometry is complicated, the fluid properties are not constant, or the boundary conditions are unsteady, the number of dimensionless parameters needed to describe a flow can become very large and the use of dimensionless equations may no longer be useful.

The dimensionless equations are useful for analytical studies and for determining the relative importance of various terms in the equations. They show, for example, that steady flow in a channel or pipe depends only on the Reynolds number; however, if the geometry changes, the flow will also be influenced by the shape of boundary. Since we are interested in computing flows in complex geometries, we shall use the dimensional form of transport equations throughout this book.

1.7 Simplified Mathematical Models

The conservation equations for mass and momentum are more complex than they appear. They are non-linear, coupled, and difficult to solve. It is difficult to prove by the existing mathematical tools that a unique solution exists for particular boundary conditions. Experience shows that the Navier-Stokes equations describe the flow of a Newtonian fluid accurately. Only in a small number of cases – mostly fully developed flows in simple geometries, e.g. in pipes, between parallel plates etc. – is it possible to obtain an analytical solution of the Navier-Stokes equations. These flows are important for studying the fundamentals of fluid dynamics, but their practical relevance is limited.

In all cases in which such a solution is possible, many terms in the equations are zero. For other flows some terms are unimportant and we may neglect them; this simplification introduces an error. In most cases, even the simplified equations cannot be solved analytically; one has to use numerical methods. The computing effort may be much smaller than for the full equations, which is a justification for simplifications. We list below some flow types for which the equations of motion can be simplified.

1.7.1 Incompressible Flow

The conservation equations for mass and momentum presented in Sects. 1.3 and 1.4 are the most general ones; they assume that all fluid and flow properties vary in space and time. In many applications the fluid density may be assumed constant. This is true not only for flows of liquids, whose compressibility may indeed be neglected, but also for gases if the Mach number is below 0.3. Such flows are said to be incompressible. If the flow is also *isothermal*, the viscosity is also constant. In that case the mass and momentum conservation equations (1.6) and (1.16) reduce to:

$$\operatorname{div} \boldsymbol{v} = 0 \;, \tag{1.32}$$

$$\frac{\partial u_i}{\partial t} + \operatorname{div}(u_i \boldsymbol{v}) = \operatorname{div}(\nu \operatorname{grad} u_i) - \frac{1}{\rho} \operatorname{div}(p\, \boldsymbol{i}_i) + b_i \;, \tag{1.33}$$

where $\nu = \mu/\rho$ is the kinematic viscosity. This simplification is generally not of a great value, as the equations are hardly any simpler to solve. However, it does help in numerical solution.

1.7.2 Inviscid (Euler) Flow

In flows far from solid surfaces, the effects of viscosity are usually very small. If viscous effects are neglected altogether, i.e. if we assume that the stress tensor reduces to $\mathsf{T} = -p\mathsf{I}$, the Navier-Stokes equations reduce to the Euler equations. The continuity equation is identical to (1.6), and the momentum equations are:

$$\frac{\partial(\rho u_i)}{\partial t} + \operatorname{div}(\rho u_i \boldsymbol{v}) = -\operatorname{div}(p\, \boldsymbol{i}_i) + \rho b_i \;. \tag{1.34}$$

Since the fluid is assumed to be inviscid, it cannot stick to walls and slip is possible at solid boundaries. The Euler equations are often used to study compressible flows at high Mach numbers. At high velocities, the Reynolds number is very high and viscous and turbulence effects are important only in a small region near the walls. These flows are often well predicted using the Euler equations.

Although the Euler equations are not easy to solve, the fact that no boundary layer near the walls need be resolved allows the use of coarser grids. Thus flows over the whole aircraft have been simulated using Euler equations; accurate resolution of the viscous region would require much more computer resource; such simulations are being done on a research basis at present.

There are many methods designed to solve compressible Euler equations. Some of them will be briefly described in Chap. 10. More details on these methods can be found in books by Hirsch (1991), Fletcher (1991) and Anderson et al. (1984), among others. The solution methods described in this book can also be used to solve the compressible Euler equations and, as we shall see in Chap. 10, they perform as well as the special methods designed for compressible flows.

1.7.3 Potential Flow

One of the simplest flow models is potential flow. The fluid is assumed to be inviscid (as in the Euler equations); however, an additional condition is imposed on the flow – the velocity field must be irrotational, i.e.:

$$\operatorname{rot} \boldsymbol{v} = 0 \ . \tag{1.35}$$

From this condition it follows that there exists a *velocity potential* Φ, such that the velocity vector can be defined as $\boldsymbol{v} = -\operatorname{grad} \Phi$. The continuity equation for an incompressible flow, $\operatorname{div} \boldsymbol{v} = 0$, then becomes a Laplace equation for the potential Φ:

$$\operatorname{div} (\operatorname{grad} \Phi) = 0 \ . \tag{1.36}$$

The momentum equation can then be integrated to give the Bernoulli equation, an algebraic equation that can be solved once the potential is known. Potential flows are therefore described by the scalar Laplace equation. The latter cannot be solved analytically for arbitrary geometries, although there are simple analytical solutions (uniform flow, source, sink, vortex), which can also be combined to create more complicated flows e.g. flow around a cylinder.

For each velocity potential Φ one can also define the corresponding *streamfunction* Ψ. The velocity vectors are tangential to streamlines (lines of constant streamfunction); the streamlines are orthogonal to lines of constant potential, so these families of lines form an orthogonal *flow net.*

Potential flows are important but not very realistic. For example, the potential theory leads to D'Alembert's paradox, i.e. a body experiences neither drag nor lift in a potential flow.

1.7.4 Creeping (Stokes) Flow

When the flow velocity is very small, the fluid is very viscous, or the geometric dimensions are very small (i.e. when the Reynolds number is small), the convective (inertial) terms in the Navier-Stokes equations play a minor role and can be neglected (see the dimensionless form of the momentum equation, Eq. (1.30)). The flow is then dominated by the viscous, pressure, and body forces and is called *creeping flow.* If the fluid properties can be considered constant, the momentum equations become linear; they are usually called *Stokes equations.* Due to the low velocities the unsteady term can also be neglected, a substantial simplification. The continuity equation is identical to Eq. (1.32), while the momentum equations become:

$$\operatorname{div} (\mu \operatorname{grad} u_i) - \frac{1}{\rho} \operatorname{div} (p \, \boldsymbol{i}_i) + b_i = 0 \ . \tag{1.37}$$

Creeping flows are found in porous media, coating technology, micro-devices etc.

1.7.5 Boussinesq Approximation

In flows accompanied by heat transfer, the fluid properties are normally functions of temperature. The variations may be small and yet be the cause of the

fluid motion. If the density variation is not large, one may treat the density as constant in the unsteady and convection terms, and treat it as variable only in the gravitational term. This is called the *Boussinesq approximation.* One usually assumes that the density varies linearly with temperature. If one includes the effect of the body force on the mean density in the pressure term as described in Sect. 1.4, the remaining term can be expressed as:

$$(\rho - \rho_0)g_i = -\rho_0 g_i \beta (T - T_0) , \tag{1.38}$$

where β is the coefficient of volumetric expansion. This approximation introduces errors of the order of 1% if the temperature differences are below e.g. 2° for water and 15° for air. The error may be more substantial when temperature differences are larger; the solution may even be qualitatively wrong (for an example, see Bückle and Perić, 1992).

1.7.6 Boundary Layer Approximation

When the flow has a predominant direction (i.e. there is no reversed flow or recirculation) and the variation of the geometry is gradual, the flow is mainly influenced by what happened upstream. Examples are flows in channels and pipes and flows over plane or mildly curved solid walls. Such flows are called *thin shear layer* or *boundary layer flows.* The Navier-Stokes equations can be simplified for such flows as follows:

- diffusive transport of momentum in the principal flow direction is much smaller than convection and can be neglected;
- the velocity component in the main flow direction is much larger than the components in other directions;
- the pressure gradient across the flow is much smaller than in the principal flow direction.

The two-dimensional boundary layer equations reduce to:

$$\frac{\partial(\rho u_1)}{\partial t} + \frac{\partial(\rho u_1 u_1)}{\partial x_1} + \frac{\partial(\rho u_2 u_1)}{\partial x_2} = \mu \frac{\partial^2 u_1}{\partial x_2^2} - \frac{\partial p}{\partial x_1} , \tag{1.39}$$

which must be solved together with the continuity equation; the equation for the momentum normal to the principal flow direction reduces to $\partial p / \partial x_2 = 0$. The pressure as a function of x_1 must be supplied by a calculation of the flow exterior to the boundary layer – which is usually assumed to be potential flow, so the boundary layer equations themselves are not a complete description of the flow. The simplified equations can be solved by using marching techniques similar to those used to solve ordinary differential equations with initial conditions. These techniques see considerable use in aerodynamics. The methods are very efficient but can be applied only to problems without separation.

1.7.7 Modeling of Complex Flow Phenomena

Many flows of practical interest are difficult to describe exactly mathematically, let alone solve exactly. These flows include turbulence, combustion, multiphase flow, and are very important. Since exact description is often impracticable, one usually uses semi-empirical models to represent these phenomena. Examples are turbulence models (which will be treated in some detail in Chap. 9), combustion models, multiphase models, etc. These models, as well as the above mentioned simplifications affect the accuracy of the solution. The errors introduced by the various approximations may either augment or cancel each other; therefore, care is needed when drawing conclusions from calculations in which models are used. Due to the importance of various kinds of errors in numerical solutions we shall devote a lot of attention to this topic. The error types will be defined and described as they are encountered.

1.8 Mathematical Classification of Flows

Quasi-linear second order partial differential equations in two independent variables can be divided into three types: hyperbolic, parabolic, and elliptic. This distinction is based on the nature of the characteristics, curves along which information about the solution is carried. Every equation of this type has two sets of characteristics.

In the hyperbolic case, the characteristics are real and distinct. This means that information propagates at finite speeds in two sets of directions. In general, the information propagation is in a particular direction so that one datum needs to be given at an initial point on each characteristic; the two sets of characteristics therefore demand two initial conditions. If there are lateral boundaries, usually only one condition is required at each point because one characteristic is carrying information out of the domain and one is carrying information in. There are, however, exceptions to this rule.

In parabolic equations the characteristics degenerate to a single real set. Consequently, only one initial condition is normally required. At lateral boundaries one condition is needed at each point.

Finally, in the elliptic case, the characteristics are imaginary or complex so there are no special directions of information propagation. Indeed, information travels essentially equally well in all directions. Generally, one boundary condition is required at each point on the boundary and the domain of solution is usually closed although part of the domain may extend to infinity. Unsteady problems are never elliptic.

These differences in the nature of the equations are reflected in the methods used to solve them. It is an important general rule that numerical methods should respect the properties of the equations they are solving.

The Navier-Stokes equations are a system of non-linear second order equations in four independent variables. Consequently the classification scheme does not apply directly to them. Nonetheless, the Navier-Stokes equations do possess many of the properties outlined above and the many of the ideas used in solving second order equations in two independent variables are applicable to them but care must be exercised.

1.8.1 Hyperbolic Flows

To begin, consider the case of unsteady inviscid compressible flow. A compressible fluid can support sound and shock waves and it is not surprising that these equations have essentially hyperbolic character. Most of the methods used to solve these equations are based on the idea that the equations are hyperbolic and, given sufficient care, they work quite well; these are the methods referred to briefly above.

For steady compressible flows, the character depends on the speed of the flow. If the flow is supersonic, the equations are hyperbolic while the equations for subsonic flow are essentially elliptic. This leads to a difficulty that we shall discuss further below.

It should be noted however, that the equations for a viscous compressible flow are still more complicated. Their character is a mixture of elements of all of the types mentioned above; they do not fit well into the classification scheme and numerical methods for them are difficult to construct.

1.8.2 Parabolic Flows

The boundary layer approximation described briefly above leads to a set of equations that have essentially parabolic character. Information travels only downstream in these equations and they may be solved using methods that are appropriate for parabolic equations.

Note, however, that the boundary layer equations require specification of a pressure that is usually obtained by solving a potential flow problem. Subsonic potential flows are governed by elliptic equations (in the incompressible limit the Laplace equation suffices) so the overall problem actually has a mixed parabolic-elliptic character.

1.8.3 Elliptic Flows

When a flow has a region of recirculation i.e. flow in a sense opposite to the principal direction of flow, information may travel upstream as well as downstream. As a result, one cannot apply conditions only at the upstream end of the flow. The problem then acquires elliptic character. This situation occurs in subsonic (including incompressible) flows and makes solution of the equations a very difficult task.

It should be noted that unsteady incompressible flows actually have a combination of elliptic and parabolic character. The former comes from the fact that information travels in both directions in space while the latter results from the fact that information can only flow forward in time. Problems of this kind are called incompletely parabolic.

1.8.4 Mixed Flow Types

As we have just seen, it is possible for a single flow to be described by equations that are not purely of one type. Another important example occurs in steady transonic flows, that is, steady compressible flows that contain both supersonic and subsonic regions. The supersonic regions are hyperbolic in character while the subsonic regions are elliptic. Consequently, it may be necessary to change the method of approximating the equations as a function of the nature of the local flow. To make matters even worse, the regions can not be determined prior to solving the equations.

1.9 Plan of This Book

This book contains twelve chapters. We now give a brief summary of the remaining eleven chapters.

In Chap. 2 an introduction to numerical solution methods is given. The advantages and disadvantages of numerical methods are discussed and the possibilities and limitations of the computational approach are outlined. This is followed by a description of the components of a numerical solution method and their properties. Finally, a brief description of basic computational methods (finite difference, finite volume and finite element) is given.

In Chap. 3 finite difference (FD) methods are described. Here we present methods of approximating first, second, and mixed derivatives, using Taylor series expansion and polynomial fitting. Derivation of higher-order methods, and treatment of non-linear terms and boundaries is discussed. Attention is also paid to the effects of grid non-uniformity on truncation error and to the estimation of discretization errors. Spectral methods are also briefly described here.

In Chap. 4 the finite volume (FV) method is described including the approximation of surface and volume integrals and the use of interpolation to obtain variable values and derivatives at locations other than cell centers. Development of higher-order schemes and simplification of the resulting algebraic equations using the deferred-correction approach is also described. Finally, implementation of the various boundary conditions is discussed.

Applications of basic FD and FV methods are described and their use is demonstrated in Chaps. 3 and 4 for structured Cartesian grids. This restriction allows us to separate the issues connected with geometric complexity

from the concepts behind discretization techniques. The treatment of complex geometries is introduced later, in Chap. 8.

In Chap. 5 we describe methods of solving the algebraic equation systems resulting from discretization. Direct methods are briefly described, but the major part of the chapter is devoted to iterative solution techniques. Incomplete lower-upper decomposition, conjugate gradients and multigrid methods are given special attention. Approaches to solving coupled and non-linear systems are also described, including the issues of under-relaxation and convergence criteria.

Chapter 6 is devoted to methods of time integration. First, the methods of solving ordinary differential equations are described, including basic methods, predictor-corrector and multipoint methods, Runge-Kutta methods. The application of these methods to the unsteady transport equations is described next, including analysis of stability and accuracy.

The complexity of the Navier-Stokes equations and special features for incompressible flows are considered in Chap. 7. The staggered and colocated variable arrangements, the pressure equation, pressure-velocity coupling and other approaches (streamfunction-vorticity, artificial compressibility, fractional step methods) are described. The solution methods for incompressible Navier-Stokes equations based on pressure-correction are described in detail for staggered and colocated Cartesian grids. Finally, some examples of two-dimensional and three-dimensional laminar flows are presented.

Chapter 8 is devoted to the treatment of complex geometries. The choices of grid type, grid properties, velocity components and variable arrangements are discussed. FD and FV methods are revisited, and the features special to complex geometries (like non-orthogonal and unstructured grids, control volumes of arbitrary shape etc.) are discussed. Special attention is paid to pressure-correction equation and boundary conditions. One section is devoted to FE methods, which are best known for their applicability to arbitrary unstructured grids.

Chapter 9 deals with computation of turbulent flows. We discuss the nature of turbulence and three methods for its simulation: direct and large-eddy simulation and methods based on Reynolds-averaged Navier-Stokes equations. Some models used in the latter two approaches are described. Examples using these approaches are presented.

In Chap. 10 compressible flows are considered. Methods designed for compressible flows are briefly discussed. The extension of the pressure-correction approach for incompressible flows to compressible flows is described. Methods for dealing with shocks (e.g. grid adaptation, total-variation-diminishing and essentially-non-oscillating schemes) are also discussed. Boundary conditions for various types of compressible flows (subsonic, transonic and supersonic) are described. Finally, examples are presented and discussed.

Chapter 11 is devoted to accuracy and efficiency improvement. The increased efficiency provided by multigrid algorithms is described first, followed

by examples. Adaptive grid methods and local grid refinement are the subject of another section. Finally, parallelization is discussed. Special attention is paid to parallel processing for implicit methods based on domain decomposition in space and time, and to analysis of the efficiency of parallel processing. Example calculations are used to demonstrate these points.

Finally, in Chap. 12 some special issues are considered. These include the treatment of moving boundaries which require moving grids and flows with free surfaces. Special effects in flows with heat and mass transfer, two phases and chemical reactions are briefly discussed.

We end this introductory chapter with a short note. Computational fluid dynamics (CFD) may be regarded as a sub-field of either fluid dynamics or numerical analysis. Competence in CFD requires that the practitioner has a fairly solid background in both areas. Poor results have been produced by individuals who are experts in one area but regarded the other as unnecessary. We hope the reader will take note of this and acquire the necessary background.

2. Introduction to Numerical Methods

2.1 Approaches to Fluid Dynamical Problems

As the first chapter stated, the equations of fluid mechanics – which have been known for over a century – are solvable for only a limited number of flows. The known solutions are extremely useful in helping to understand fluid flow but rarely can they be used directly in engineering analysis or design. The engineer has traditionally been forced to use other approaches.

In the most common approach, simplifications of the equations are used. These are usually based on a combination of approximations and dimensional analysis; empirical input is almost always required. For example, dimensional analysis shows that the drag force on an object can be represented by:

$$F_{\mathrm{D}} = C_{\mathrm{D}} S \rho v^2 , \tag{2.1}$$

where S is the frontal area presented to the flow by the body, v is the flow velocity and ρ is the density of the fluid; the parameter C_{D} is called the drag coefficient. It is a function of the other non-dimensional parameters of the problem and is nearly always obtained by correlating experimental data. This approach is very successful when the system can be described by one or two parameters so application to complex geometries (which can only be described by many parameters) are ruled out.

A related approach is arrived at by noting that for many flows non-dimensionalization of the Navier-Stokes equations leaves the Reynolds number as the only independent parameter. If the body shape is held fixed, one can get the desired results from an experiment on a scale model with that shape. The desired Reynolds number is achieved by careful selection of the fluid and the flow parameters or by extrapolation in Reynolds number; the latter can be dangerous. These approaches are very valuable and are the primary methods of practical engineering design even today.

The problem is that many flows require several dimensionless parameters for their specification and it may be impossible to set up an experiment which correctly scales the actual flow. Examples are flows around aircraft or ships. In order to achieve the same Reynolds number with smaller models, fluid velocity has to be increased. For aircraft, this may give too high a Mach number if the same fluid (air) is used; one tries to find a fluid which

allows matching of both parameters. For ships, the issue is to match both the Reynolds and Froude numbers, which is nearly impossible.

In other cases, experiments are very difficult if not impossible. For example, the measuring equipment might disturb the flow or the flow may be inaccessible (e.g. flow of a liquid silicon in a crystal growth apparatus). Some quantities are simply not measurable with present techniques or can be measured only with an insufficient accuracy.

Experiments are an efficient means of measuring global parameters, like the drag, lift, pressure drop, or heat transfer coefficients. In many cases, details are important; it may be essential to know whether flow separation occurs or whether the wall temperature exceeds some limit. As technological improvement and competition require more careful optimization of designs or, when new high-technology applications demand prediction of flows for which the database is insufficient, experimental development may be too costly and/or time consuming. Finding a reasonable alternative is essential.

An alternative – or at least a complementary method – came with the birth of electronic computers. Although many of the key ideas for numerical solution methods for partial differential equations were established more than a century ago, they were of little use before computers appeared. The performance-to-cost ratio of computers has increased at a spectacular rate since the 1950s and shows no sign of slowing down. While the first computers built in the 1950s performed only a few hundred operations per second, machines are now being designed to produce teraflops – 10^{12} floating point operations per second. The ability to store data has also increased dramatically: hard discs with ten gigabyte (10^{10} bytes or characters) capacity could be found only on supercomputers a decade ago – now they are found in personal computers. A machine that cost millions of dollars, filled a large room, and required a permanent maintenance and operating staff is now available on a desktop. It is difficult to predict what will happen in the future, but further increases in both computing speed and memory of affordable computers are certain.

It requires little imagination to see that computers might make the study of fluid flow easier and more effective. Once the power of computers had been recognized, interest in numerical techniques increased dramatically. Solution of the equations of fluid mechanics on computers has become so important that it now occupies the attention of perhaps a third of all researchers in fluid mechanics and the proportion is still increasing. This field is known as *computational fluid dynamics* (CFD). Contained within it are many subspecialties. We shall discuss only a small subset of methods for solving the equations describing fluid flow and related phenomena.

2.2 What is CFD?

As we have seen in Chap. 1, flows and related phenomena can be described by partial differential (or integro-differential) equations, which cannot be solved analytically except in special cases. To obtain an approximate solution numerically, we have to use a *discretization method* which approximates the differential equations by a system of algebraic equations, which can then be solved on a computer. The approximations are applied to small domains in space and/or time so the numerical solution provides results at *discrete locations* in space and time. Much as the accuracy of experimental data depends on the quality of the tools used, the accuracy of numerical solutions is dependent on the quality of discretizations used.

Contained within the broad field of computational fluid dynamics are activities that cover the range from the automation of well-established engineering design methods to the use of detailed solutions of the Navier-Stokes equations as substitutes for experimental research into the nature of complex flows. At one end, one can purchase design packages for pipe systems that solve problems in a few seconds or minutes on personal computers or workstations. On the other, there are codes that may require hundreds of hours on the largest super-computers. The range is as large as the field of fluid mechanics itself, making it impossible to cover all of CFD in a single work. Also, the field is evolving so rapidly that we run the risk of becoming out of date in a short time.

We shall not deal with automated simple methods in this book. The basis for them is covered in elementary textbooks and undergraduate courses and the available program packages are relatively easy to understand and to use.

We shall be concerned with methods designed to solve the equations of fluid motion in two or three dimensions. These are the methods used in non-standard applications, by which we mean applications for which solutions (or, at least, good approximations) cannot be found in textbooks or handbooks. While these methods have been used in high-technology engineering (for example, aeronautics and astronautics) from the very beginning, they are being used more frequently in fields of engineering where the geometry is complicated or some important feature (such as the prediction of the concentration of a pollutant) cannot be dealt with by standard methods. CFD is finding its way into process, chemical, civil, and environmental engineering. Optimization in these areas can produce large savings in equipment and energy costs and in reduction of environmental pollution.

2.3 Possibilities and Limitations of Numerical Methods

We have already noted some problems associated with experimental work. Some of these problems are easily dealt with in CFD. For example, if we want to simulate the flow around a moving car in a wind tunnel, we need to fix

the car model and blow air at it – but the floor has to move at the air speed, which is difficult to do. It is not difficult to do in a numerical simulation. Other types of boundary conditions are easily prescribed in computations; for example, temperature or opaqueness of the fluid pose no problem. *If* we solve the unsteady three-dimensional Navier-Stokes equations accurately (as in direct simulation of turbulence), we obtain a complete data set from which any quantity of physical significance can be derived.

This sounds to good to be true. Indeed, these advantages of CFD are conditional on being able to solve the Navier-Stokes equations accurately, which is extremely difficult for most flows of engineering interest. We shall see in Chap. 9 why obtaining accurate numerical solutions of the Navier-Stokes equations for high Reynolds number flows is so difficult.

If we are unable to obtain *accurate* solutions for all flows, we have to determine what we can produce and learn to analyze and judge the results. First of all, we have to bear in mind that numerical results are always *approximate.* There are reasons for differences between computed results and 'reality' i.e. errors arise from each part of the process used to produce numerical solutions:

- The differential equations may contain approximations or idealizations, as discussed in Sect. 1.7;
- Approximations are made in the discretization process;
- In solving the discretized equations, iterative methods are used. Unless they are run for a very long time, the exact solution of the discretized equations is not produced.

When the governing equations are known accurately (e.g. the Navier-Stokes equations for incompressible Newtonian fluids), solutions of any desired accuracy can be achieved in principle. However, for many phenomena (e.g. turbulence, combustion, and multiphase flow) the exact equations are either not available or numerical solution is not feasible. This makes introduction of models a necessity. Even if we solve the equations exactly, the solution would not be a correct representation of reality. In order to *validate* the models, we have to rely on experimental data. Even when the exact treatment is possible, models are often needed to reduce the cost.

Discretization errors can be reduced by using more accurate interpolation or approximations or by applying the approximations to smaller regions but this usually increases the time and cost of obtaining the solution. Compromise is usually needed. We shall present some schemes in detail but shall also point out ways of creating more accurate approximations.

Compromises are also needed in solving the discretized equations. Direct solvers, which obtain accurate solutions, are seldom used, because they are too costly. Iterative methods are more common but the errors due to stopping the iteration process too soon need to be taken into account.

Errors and their estimation will be emphasized throughout this book. We shall present error estimates for many examples; the need to analyze and estimate numerical errors can not be overemphasized.

Visualization of numerical solutions using vector, contour or other kinds of plots or movies (videos) of unsteady flows is important for the interpretation of results. It is far and away the most effective means of interpreting the huge amount of data produced by a calculation. However, there is the danger that an erroneous solution may look good but may not correspond to the actual boundary conditions, fluid properties etc.! The authors have encountered incorrect numerically produced flow features that could be and have been interpreted as physical phenomena. Industrial users of commercial CFD codes should especially be careful, as the optimism of salesmen is legendary. Wonderful color pictures make a great impression but are of no value if they are not quantitatively correct. Results must be examined very critically before they are believed.

2.4 Components of a Numerical Solution Method

Since this book is meant not only for users of commercial codes but also for young researchers developing new codes, we shall present the important ingredients of a numerical solution method here. More details will be presented in the following chapters.

2.4.1 Mathematical Model

The starting point of any numerical method is the mathematical model, i.e. the set of partial differential or integro-differential equations and boundary conditions. Some sets of equations used for flow prediction were presented in Chap. 1. One chooses an appropriate model for the target application (incompressible, inviscid, turbulent; two- or three-dimensional, etc.). As already mentioned, this model may include simplifications of the exact conservation laws. A solution method is usually designed for a particular set of equations. Trying to produce a general purpose solution method, i.e. one which is applicable to all flows, is impractical, if not impossible and, as with most general purpose tools, they are usually not optimum for any one application.

2.4.2 Discretization Method

After selecting the mathematical model, one has to choose a suitable discretization method, i.e. a method of approximating the differential equations by a system of algebraic equations for the variables at some set of discrete locations in space and time. There are many approaches, but the most important of which are: finite difference (FD), finite volume (FV) and finite element (FE) methods. Important features of these three kinds of discretization methods are described at the end of this chapter. Other methods, like spectral schemes, boundary element methods, and cellular automata are used in CFD but their use is limited to special classes of problems.

Each type of method yields the same solution if the grid is very fine. However, some methods are more suitable to some classes of problems than others. The preference is often determined by the attitude of the developer. We shall discuss the pros and cons of the various methods later.

2.4.3 Coordinate and Basis Vector Systems

It was mentioned in Chap. 1 that the conservation equations can be written in many different forms, depending on the coordinate system and the basis vectors used. For example one can select Cartesian, cylindrical, spherical, curvilinear orthogonal or non-orthogonal coordinate systems, which may be fixed or moving. The choice depends on the target flow, and may influence the discretization method and grid type to be used.

One also has to select the basis in which vectors and tensors will be defined (fixed or variable, covariant or contravariant, etc.). Depending on this choice, the velocity vector and stress tensor can be expressed in terms of e.g. Cartesian, covariant or contravariant, physical or non-physical coordinate-oriented components. In this book we shall use Cartesian components exclusively for reasons explained in Chap. 8.

2.4.4 Numerical Grid

The discrete locations at which the variables are to be calculated are defined by the numerical grid which is essentially a discrete representation of the geometric domain on which the problem is to be solved. It divides the solution domain into a finite number of subdomains (elements, control volumes etc.). Some of the options available are the following:

- *Structured (regular) grid* — Regular or structured grids consist of families of grid lines with the property that members of a single family do not cross each other and cross each member of the other families only once. This allows the lines of a given set to be numbered consecutively. The position of any grid point (or control volume) within the domain is uniquely identified by a set of two (in 2D) or three (in 3D) indices, e.g. (i, j, k).
  This is the simplest grid structure, since it is logically equivalent to a Cartesian grid. Each point has four nearest neighbors in two dimensions and six in three dimensions; one of the indices of each neighbor of point P (indices i, j, k) differs by ± 1 from the corresponding index of P. An example of a structured 2D grid is shown in Fig. 2.1. This neighbor connectivity simplifies programming and the matrix of the algebraic equation system has a regular structure, which can be exploited in developing a solution technique. Indeed, there is a large number of efficient solvers applicable only to structured grids (see Chap. 5). The disadvantage of structured grids is that they can be used only for geometrically simple solution domains. Another disadvantage is that it may be difficult to control the distribution of the

grid points: concentration of points in one region for reasons of accuracy produces unnecessarily small spacing in other parts of the solution domain and a waste of resources. This problem is exaggerated in 3D problems. The long thin cells may also affect the convergence adversely.
Structured grids may be of H-, O-, or C-type; the names are derived from the shapes of the grid lines. Figure 2.1 shows an H-type grid which, when mapped onto a rectangle, has distinct east, west, north, and south boundaries. Figure 2.3 shows an O-type structured grid around a cylinder. In this type of grid, one set of grid lines is "endless"; if the grid lines are treated as coordinate lines and we follow the coordinate around the cylinder, it will continuously increase and, to avoid a problem, one must introduce an artificial "cut" at which the coordinate jumps from a finite value to zero. At the cut, the grid can be "unwrapped" but the neighboring points must be treated as interior grid points, in contrast to the treatment applied at the boundaries of an H-type grid. The outer grid in Fig. 2.3 is again of H-type. The block grid around the hydrofoil in Fig. bloknmgr is of C-type. In this type of grid, points on portions of one grid line coincide, requiring the introduction of a cut similar to the ones found in O-type grids. This type of grid is often used for bodies with sharp edges for which they are capable of good grid quality.
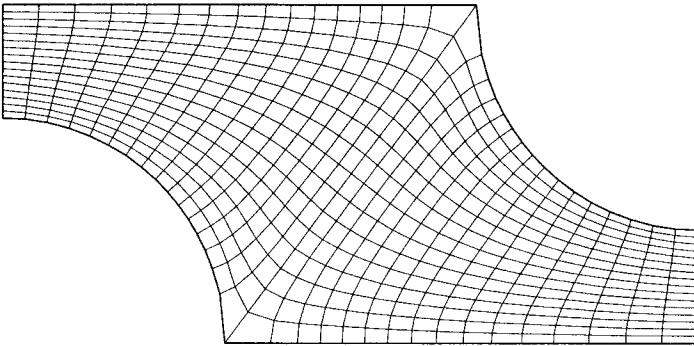
Fig. 2.1. Example of a 2D, structured, non-orthogonal grid, designed for calculation of flow in a symmetry segment of a staggered tube bank

- *Block-structured grid* — In a block structured grid, there is a two (or more) level subdivision of solution domain. On the coarse level, there are blocks which are relatively large segments of the domain; their structure may be irregular and they may or may not overlap. On the fine level (within each block) a structured grid is defined. Special treatment is necessary at block interfaces. Some methods of this kind are described in Chap. 8.
  In Fig. 2.2 a block-structured grid with matching at the interfaces is shown; it is designed for the calculation of 2D flow around a cylinder in a channel and contains three blocks.
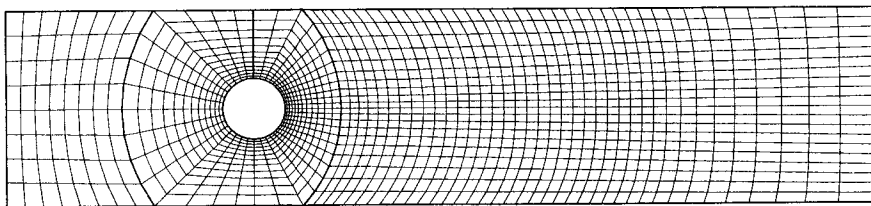
Fig. 2.2. Example of a 2D block-structured grid which matches at interfaces, used to calculate flow around a cylinder in a channel

In Fig. 2.3 a block-structured grid with non-matching interfaces is shown; it was used to calculate the flow around a submerged hydrofoil. It consists of five blocks of grids of different fineness. This kind of grid is more flexible than the previous ones, as it allows use of finer grids in regions, where greater resolution is required. The non-matching interface can be treated in a fully conservative manner, as will be discussed in Chap. 8. The programming is more difficult than for grid types described above. Solvers for structured grids can be applied block-wise, and complex flow domains can be treated with these grids. Local refinement is possible block-wise (i.e., the grid may be refined in some blocks).
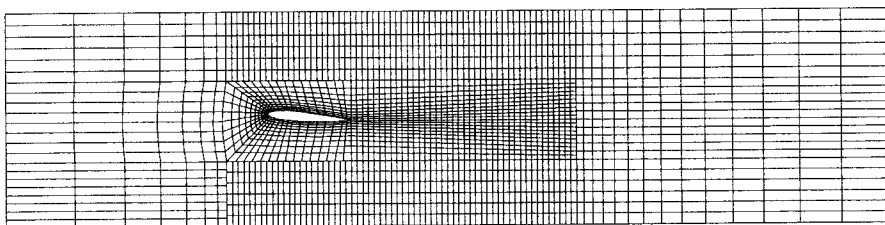
Fig. 2.3. Example of a 2D block-structured grid which does not match at interfaces, designed for calculation of flow around a hydrofoil under a water surface

Block-structured grids with overlapping blocks are sometimes called *composite* or *Chimera* grids. One such grid is shown in Fig. 2.4. In the overlap region, boundary conditions for one block are obtained by interpolating the solution from the other (overlapped) block. The disadvantage of these grids is that conservation is not easily enforced at block boundaries. The advantages of this approach are that complex domains are dealt with more easily and it can be used to follow moving bodies: one block is attached to the body and moves with it, while a stagnant grid covers the surroundings. This type of grid is not very often used, although it has strong supporters (Tu and Fuchs, 1992; Perng and Street, 1991; Hinatsu and Ferziger, 1991; Zang and Street, 1995; Hubbard and Chen, 1994, 1995).
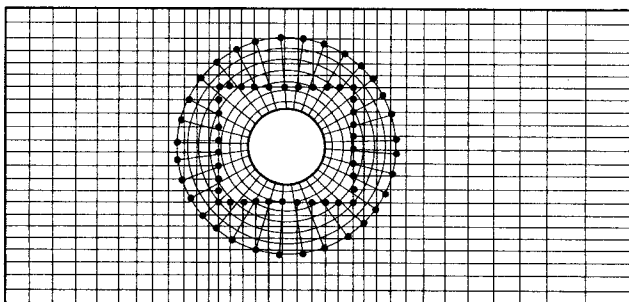
Fig. 2.4. A composite 2D grid, used to calculate flow around a cylinder in a channel

- *Unstructured grids* — For very complex geometries, the most flexible type of grid is one which can fit an arbitrary solution domain boundary. In principle, such grids could be used with any discretization scheme, but they are best adapted to the finite volume and finite element approaches. The elements or control volumes may have any shape; nor is there a restriction on the number of neighbor elements or nodes. In practice, grids made of triangles or quadrilaterals in 2D, and tetrahedra or hexahedra in 3D are most often used. Such grids can be generated automatically by existing algorithms. If desired, the grid can be made orthogonal, the aspect ratio is easily controlled, and the grid may be easily locally refined. The advantage of flexibility is offset by the disadvantage of the irregularity of the data structure. Node locations and neighbor connections need be specified explicitly. The matrix of the algebraic equation system no longer has regular, diagonal structure; the band width needs to be reduced by reordering of the points. The solvers for the algebraic equation systems are usually slower than those for regular grids.
  Unstructured grids are usually used with finite element methods and, increasingly, with finite volume methods. Computer codes for unstructured grids are more flexible. They need not be changed when the grid is locally refined, or when elements or control volumes of different shapes are used. However, grid generation and pre-processing are usually much more difficult. The finite volume method presented in this book is applicable to unstructured grids. An example of an unstructured grid is shown in Fig. 2.5.

Methods of grid generation will not be covered in detail in this book. Grid properties and some basic grid generation methods are discussed briefly in Chap. 8; there is a vast literature devoted to grid generation and interested reader is referred to books by Thompson et al. (1985) and Arcilla et al. (1991).
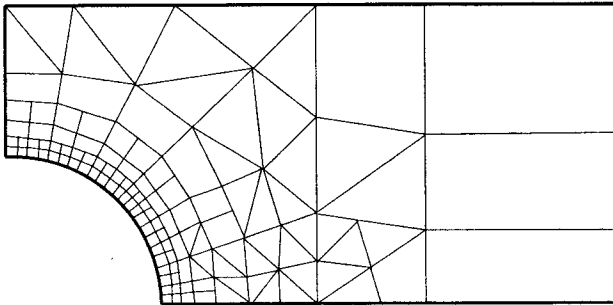
Fig. 2.5. Example of a 2D unstructured grid

2.4.5 Finite Approximations

Following the choice of grid type, one has to select the approximations to be used in the discretization process. In a finite difference method, approximations for the derivatives at the grid points have to be selected. In a finite volume method, one has to select the methods of approximating surface and volume integrals. In a finite element method, one has to choose the shape functions (elements) and weighting functions.

There are many possibilities to choose from; some of those most often used are presented in this book, some are simply mentioned and many more can be created. The choice influences the accuracy of the approximation. It also affects the difficulty of developing the solution method, coding it, debugging it, and the speed of the code. More accurate approximations involve more nodes and give fuller coefficient matrices. The increased memory requirement may require using coarser grids, partially offsetting the advantage of higher accuracy. A compromise between simplicity, ease of implementation, accuracy and computational efficiency has to be made. The second-order methods presented in this book were selected with this compromise in mind.

2.4.6 Solution Method

Discretization yields a large system of non-linear algebraic equations. The method of solution depends on the problem. For unsteady flows, methods based on those used for initial value problems for ordinary differential equations (marching in time) are used. At each time step an elliptic problem has to be solved. Steady flow problems are usually solved by pseudo-time marching or an equivalent iteration scheme. Since the equations are non-linear, an iteration scheme is used to solve them. These methods use successive linearization of the equations and the resulting linear systems are almost always solved by iterative techniques. The choice of solver depends on the grid type and the number of nodes involved in each algebraic equation. Some solvers will be presented in Chap. 5.

2.4.7 Convergence Criteria

Finally, one needs to set the convergence criteria for the iterative method. Usually, there are two levels of iterations: inner iterations, within which the linear equation are solved, and outer iterations, that deal with the non-linearity and coupling of the equations. Deciding when to stop the iterative process on each level is important, from both the accuracy and efficiency points of view. These issues are dealt with in Chaps. 5 and 11.

2.5 Properties of Numerical Solution Methods

The solution method should have certain properties. In most cases, it is not possible to analyze the complete solution method. One analyzes the components of the method; if the components do not possess the desired properties, neither will the complete method but the reverse is not necessarily true. The most important properties are summarized below.

2.5.1 Consistency

The discretization should become exact as the grid spacing tends to zero. The difference between the discretized equation and the exact one is called the *truncation error*. It is usually estimated by replacing all the nodal values in the discrete approximation by a Taylor series expansion about a single point. As a result one recovers the original differential equation plus a remainder, which represents the truncation error. For a method to be *consistent*, the truncation error must become zero when the mesh spacing $\Delta t \rightarrow 0$ and/or $\Delta x_i \rightarrow 0$. Truncation error is usually proportional to a power of the grid spacing Δx_i and/or the time step Δt. If the most important term is proportional to $(\Delta x)^n$ or $(\Delta t)^n$ we call the method an nth-order approximation; $n > 0$ is required for consistency. Ideally, all terms should be discretized with approximations of the same order of accuracy; however, some terms (e.g. convective terms in high Reynolds number flows or diffusive terms in low Reynolds number flows) may be dominant in a particular flow and it may be reasonable to treat them with more accuracy than the others.

Some discretization methods lead to truncation errors which are functions of the ratio of Δx_i to Δt or vice versa. In such a case the consistency requirement is only conditionally fulfilled: Δx_i and Δt must be reduced in a way that allows the appropriate ratio to go to zero. In the next two chapters we shall demonstrate consistency for several discretization schemes.

Even if the approximations are consistent, it does not necessarily mean that the solution of the discretized equation system will become the exact solution of the differential equation in the limit of small step size. For this to happen, the solution method has to be *stable*; this is defined below.

2.5.2 Stability

A numerical solution method is said to be stable if it does not magnify the errors that appear in the course of numerical solution process. For temporal problems, stability guarantees that the method produces a bounded solution whenever the solution of the exact equation is bounded. For iterative methods, a stable method is one that does not diverge. Stability can be difficult to investigate, especially when boundary conditions and non-linearities are present. For this reason, it is common to investigate the stability of a method for linear problems with constant coefficients without boundary conditions. Experience shows that the results obtained in this way can often be applied to more complex problems but there are notable exceptions.

The most widely used approach to studying stability of numerical schemes is the von Neumann's method. We shall describe it briefly for one scheme in Chap. 6. Most of the schemes to be described in this book have been analyzed for stability and we shall state the important result when describing each scheme. However, when solving complicated, non-linear and coupled equations with complicated boundary conditions, there are few stability results so we may have to rely on experience and intuition. Many solution schemes require that the time step be smaller than a certain limit or that under-relaxation be used. We shall discuss these issues and give guidelines for selecting time step size and values of under-relaxation parameters in Chaps. 6 and 7.

2.5.3 Convergence

A numerical method is said to be convergent if the solution of the discretized equations tends to the exact solution of the differential equation as the grid spacing tends to zero. For linear initial value problems, the *Lax equivalence theorem* (Richtmyer and Morton, 1967) states that "given a properly posed linear initial value problem and a finite difference approximation to it that satisfies the consistency condition, stability is the necessary and sufficient condition for convergence". Obviously, a consistent scheme is useless unless the solution method converges.

For non-linear problems which are strongly influenced by boundary conditions, the stability and convergence of a method are difficult to demonstrate. Therefore convergence is usually checked using numerical experiments, i.e. repeating the calculation on a series of successively refined grids. If the method is stable and if all approximations used in the discretization process are consistent, we usually find that the solution does converge to a *grid-independent solution.* For sufficiently small grid sizes, the rate of convergence is governed by the order of principal truncation error component. This allows us to estimate the error in the solution. We shall describe this in detail in Chaps. 3 and 5.

2.5.4 Conservation

Since the equations to be solved are conservation laws, the numerical scheme should also – on both a local and a global basis – respect these laws. This means that, at steady state and in the absence of sources, the amount of a conserved quantity leaving a closed volume is equal to the amount entering that volume. If the strong conservation form of equations and a finite volume method are used, this is guaranteed for each individual control volume and for the solution domain as a whole. Other discretization methods can be made conservative if care is taken in the choice of approximations. The treatment of sources or sink terms should be consistent so that the total source or sink in the domain is equal to the net flux of the conserved quantity through the boundaries.

This is an important property of the solution method, since it imposes a constraint on the solution error. If the conservation of mass, momentum and energy are insured, the error can only improperly distribute these quantities over the solution domain. Non-conservative schemes can produce artificial sources and sinks, changing the balance both locally and globally. However, non-conservative schemes can be consistent and stable and therefore lead to correct solutions in the limit of very fine grids. The errors due to non-conservation are in most cases appreciable only on relatively coarse grids. The problem is that it is difficult to know on which grid are these errors small enough. Conservative schemes are therefore preferred.

2.5.5 Boundedness

Numerical solutions should lie within proper bounds. Physically non-negative quantities (like density, kinetic energy of turbulence) must always be positive; other quantities, such as concentration, must lie between 0% and 100%. In the absence of sources, some equations (e.g. the heat equation for the temperature when no heat sources are present) require that the minimum and maximum values of the variable be found on the boundaries of the domain. These conditions should be inherited by the numerical approximation.

Boundedness is difficult to guarantee. We shall show later on that only some first order schemes guarantee this property. All higher-order schemes can produce unbounded solutions; fortunately, this usually happens only on grids that are too coarse, so a solution with undershoots and overshoots is usually an indication that the errors in the solution are large and the grid needs some refinement (at least locally). The problem is that schemes prone to producing unbounded solutions may have stability and convergence problems. These methods should be avoided, if possible.

2.5.6 Realizability

Models of phenomena which are too complex to treat directly (for example, turbulence, combustion, or multiphase flow) should be designed to guarantee

physically realistic solutions. This is not a numerical issue *per se* but models that are not realizable may result in unphysical solutions or cause numerical methods to diverge. We shall not deal with these issues in this book, but if one wants to implement a model in a CFD code, one has to be careful about this property.

2.5.7 Accuracy

Numerical solutions of fluid flow and heat transfer problems are only *approximate solutions*. In addition to the errors that might be introduced in the course of the development of the solution algorithm, in programming or setting up the boundary conditions, numerical solutions always include three kinds of systematic errors:

- *Modeling errors*, which are defined as the difference between the actual flow and the exact solution of the mathematical model;
- *Discretization errors*, defined as the difference between the exact solution of the conservation equations and the exact solution of the algebraic system of equations obtained by discretizing these equations, and
- *Iteration errors*, defined as the difference between the iterative and exact solutions of the algebraic equations systems.

Iteration errors are often called *convergence errors* (which was the case in the earlier editions of this book). However, the term *convergence* is used not only in conjunction with error reduction in iterative solution methods, but is also (quite appropriately) often associated with the convergence of numerical solutions towards a grid-independent solution, in which case it is closely linked to discretization error. To avoid confusion, we shall adhere to the above definition of errors and, when discussing issues of convergence, always indicate which type of convergence we are talking about.

It is important to be aware of the existence of these errors, and even more to try to distinguish one from another. Various errors may cancel each other, so that sometimes a solution obtained on a coarse grid may agree better with the experiment than a solution on a finer grid – which, by definition, should be more accurate.

Modeling errors depend on the assumptions made in deriving the transport equations for the variables. They may be considered negligible when laminar flows are investigated, since the Navier-Stokes equations represent a sufficiently accurate model of the flow. However, for turbulent flows, two-phase flows, combustion etc., the modeling errors may be very large – the exact solution of the model equations may be *qualitatively* wrong. Modeling errors are also introduced by simplifying the geometry of the solution domain, by simplifying boundary conditions etc. These errors are not known *a priori*; they can only be evaluated by comparing solutions in which the discretization and convergence errors are negligible with accurate experimental data or with data obtained by more accurate models (e.g. data from direct

simulation of turbulence, etc.). It is essential to control and estimate the convergence and discretization errors before the models of physical phenomena (like turbulence models) can be judged.

We mentioned above that discretization approximations introduce errors which decrease as the grid is refined, and that the order of the approximation is a measure of accuracy. However, on a given grid, methods of the same order may produce solution errors which differ by as much as an order of magnitude. This is because the order only tells us the *rate* at which the error decreases as the mesh spacing is reduced – it gives no information about the error on a single grid. We shall show how discretization errors can be estimated in the next chapter.

Errors due to iterative solution and round-off are easier to control; we shall see how this can be done in Chap. 5, where iterative solution methods are introduced.

There are many solution schemes and the developer of a CFD code may have a difficult time deciding which one to adopt. The ultimate goal is to obtain desired accuracy with least effort, or the maximum accuracy with the available resources. Each time we describe a particular scheme we shall point out its advantages or disadvantages with respect to these criteria.

2.6 Discretization Approaches

2.6.1 Finite Difference Method

This is the oldest method for numerical solution of PDE's, believed to have been introduced by Euler in the 18th century. It is also the easiest method to use for simple geometries.

The starting point is the conservation equation in differential form. The solution domain is covered by a grid. At each grid point, the differential equation is approximated by replacing the partial derivatives by approximations in terms of the nodal values of the functions. The result is one algebraic equation per grid node, in which the variable value at that and a certain number of neighbor nodes appear as unknowns.

In principle, the FD method can be applied to any grid type. However, in all applications of the FD method known to the authors, it has been applied to structured grids. The grid lines serve as local coordinate lines.

Taylor series expansion or polynomial fitting is used to obtain approximations to the first and second derivatives of the variables with respect to the coordinates. When necessary, these methods are also used to obtain variable values at locations other than grid nodes (interpolation). The most widely used methods of approximating derivatives by finite differences are described in Chap. 3.

On structured grids, the FD method is very simple and effective. It is especially easy to obtain higher-order schemes on regular grids; some will be

mentioned in Chap. 3. The disadvantage of FD methods is that the conservation is not enforced unless special care is taken. Also, the restriction to simple geometries is a significant disadvantage in complex flows.

2.6.2 Finite Volume Method

The FV method uses the integral form of the conservation equations as its starting point. The solution domain is subdivided into a finite number of contiguous control volumes (CVs), and the conservation equations are applied to each CV. At the centroid of each CV lies a computational node at which the variable values are to be calculated. Interpolation is used to express variable values at the CV surface in terms of the nodal (CV-center) values. Surface and volume integrals are approximated using suitable quadrature formulae. As a result, one obtains an algebraic equation for each CV, in which a number of neighbor nodal values appear.

The FV method can accommodate any type of grid, so it is suitable for complex geometries. The grid defines only the control volume boundaries and need not be related to a coordinate system. The method is conservative by construction, so long as surface integrals (which represent convective and diffusive fluxes) are the same for the CVs sharing the boundary.

The FV approach is perhaps the simplest to understand and to program. All terms that need be approximated have physical meaning which is why it is popular with engineers.

The disadvantage of FV methods compared to FD schemes is that methods of order higher than second are more difficult to develop in 3D. This is due to the fact that the FV approach requires three levels of approximation: interpolation, differentiation, and integration. We shall give a detailed description of the FV method in Chap. 4; it is the most used method in this book.

2.6.3 Finite Element Method

The FE method is similar to the FV method in many ways. The domain is broken into a set of discrete volumes or finite elements that are generally unstructured; in 2D, they are usually triangles or quadrilaterals, while in 3D tetrahedra or hexahedra are most often used. The distinguishing feature of FE methods is that the equations are multiplied by a *weight function* before they are integrated over the entire domain. In the simplest FE methods, the solution is approximated by a linear shape function within each element in a way that guarantees continuity of the solution across element boundaries. Such a function can be constructed from its values at the corners of the elements. The weight function is usually of the same form.

This approximation is then substituted into the weighted integral of the conservation law and the equations to be solved are derived by requiring the

derivative of the integral with respect to each nodal value to be zero; this corresponds to selecting the best solution within the set of allowed functions (the one with minimum residual). The result is a set of non-linear algebraic equations.

An important advantage of finite element methods is the ability to deal with arbitrary geometries; there is an extensive literature devoted to the construction of grids for finite element methods. The grids are easily refined; each element is simply subdivided. Finite element methods are relatively easy to analyze mathematically and can be shown to have optimality properties for certain types of equations. The principal drawback, which is shared by any method that uses unstructured grids, is that the matrices of the linearized equations are not as well structured as those for regular grids making it more difficult to find efficient solution methods. For more details on finite element methods and their application to the Navier-Stokes equations, see books by Oden (1972), Zinkiewicz (1977), Chung (1978), Baker (1983), Girault and Raviart (1986) or Fletcher (1991).

A hybrid method called *control-volume-based finite element method* (CV-FEM) should also be mentioned. In it, shape functions are used to describe the variation of the variables over an element. Control volumes are formed around each node by joining the centroids of the elements. The conservation equations in integral form are applied to these CVs in the same way as in the finite volume method. The fluxes through CV boundaries and the source terms are calculated element-wise. We shall give a short description of this approach in Chap. 8.

3. Finite Difference Methods

3.1 Introduction

As was mentioned in Chap. 1, all conservation equations have similar structure and may be regarded as special cases of a generic transport equation, Eq. (1.26), (1.27) or (1.28). For this reason, we shall treat only a single, generic conservation equation in this and the following chapters. It will be used to demonstrate discretization methods for the terms which are common to all conservation equations (convection, diffusion, and sources). The special features of the Navier-Stokes equations and techniques for solving coupled non-linear problems will be introduced later. Also, for the time being, the unsteady term will be dropped so we consider only time-independent problems.

For simplicity, we shall use only Cartesian grids at this point. The equation we shall deal with is:

$$\frac{\partial(\rho u_j \phi)}{\partial x_j} = \frac{\partial}{\partial x_j}\left(\Gamma \frac{\partial \phi}{\partial x_j}\right) + q_\phi \; . \tag{3.1}$$

We shall assume that ρ, u_j, Γ and q_ϕ are known. This may not be the case because the velocity may not have been computed yet and the properties of the fluid may depend on the temperature and, if turbulence models are used, on the velocity field as well. As we shall see, the iterative schemes used to solve these equations treat ϕ as the only unknown; all other variables are fixed at their values determined on the previous iteration so regarding these as known is a reasonable approach.

The special features of non-orthogonal and unstructured grids will be discussed in Chap. 8. Furthermore, of the many possible discretization techniques, only a selected few which illustrate the main ideas will be described; others may be found in the literature cited.

3.2 Basic Concept

The first step in obtaining a numerical solution is to discretize the geometric domain – i.e. a numerical grid must be defined. In finite difference (FD)

discretization methods the grid is usually locally structured, i.e. each grid node may be considered the origin of a local coordinate system, whose axes coincide with grid lines. This also implies that two grid lines belonging to the same family, say ξ_1, do not intersect, and that any pair of grid lines belonging to different families, say $\xi_1 =$ const. and $\xi_2 =$ const., intersect only once. In three dimensions, three grid lines intersect at each node; none of these lines intersect each other at any other point. Figure 3.1 shows examples of one-dimensional (1D) and two-dimensional (2D) Cartesian grids used in FD methods.
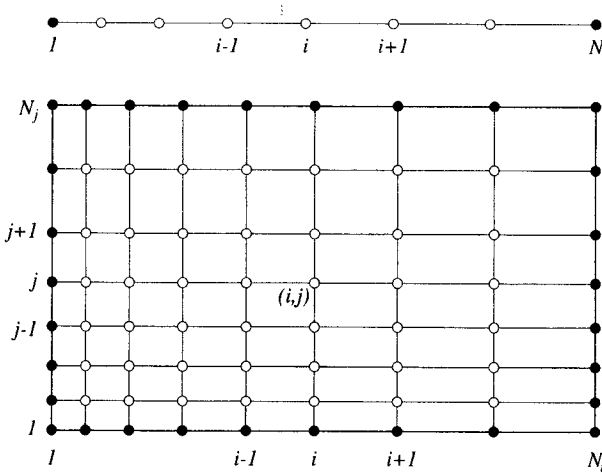
Fig. 3.1. An example of a 1D (above) and 2D (below) Cartesian grid for FD methods (full symbols denote boundary nodes and open symbols denote computational nodes)

Each node is uniquely identified by a set of indices, which are the indices of the grid lines that intersect at it, (i, j) in 2D and (i, j, k) in 3D. The neighbor nodes are defined by increasing or reducing one of the indices by unity.

The generic scalar conservation equation in differential form, (3.1), serves as the starting point for FD methods. As it is linear in ϕ, it will be approximated by a system of linear algebraic equations, in which the variable values at the grid nodes are the unknowns. The solution of this system approximates the solution to the partial differential equation (PDE).

Each node thus has one unknown variable value associated with it and must provide one algebraic equation. The latter is a relation between the variable value at that node and those at some of the neighboring nodes. It is obtained by replacing each term of the PDE at the particular node by a finite-difference approximation. Of course, the numbers of equations and unknowns must be equal. At boundary nodes where variable values are given (Dirichlet conditions), no equation is needed. When the boundary conditions

involve derivatives (as in Neumann conditions), the boundary condition must be discretized to contribute an equation to the set that must be solved.

The idea behind finite difference approximations is borrowed directly from the definition of a derivative:

$$\left(\frac{\partial \phi}{\partial x}\right)_{x_i} = \lim_{\Delta x \to 0} \frac{\phi(x_i + \Delta x) - \phi(x_i)}{\Delta x} . \tag{3.2}$$

A geometrical interpretation is shown in Fig. 3.2 to which we shall refer frequently. The first derivative $\partial \phi / \partial x$ at a point is the slope of the tangent to the curve $\phi(x)$ at that point, the line marked 'exact' in the figure. Its slope can be approximated by the slope of a line passing through two nearby points on the curve. The dotted line shows approximation by a *forward difference*; the derivative at x_i is approximated by the slope of a line passing through the point x_i and another point at $x_i + \Delta x$. The dashed line illustrates approximation by *backward difference*: for which the second point is $x_i - \Delta x$. The line labeled 'central' represents approximation by a *central difference*: it uses the slope of a line passing through two points lying on opposite sides of the point at which the derivative is approximated.
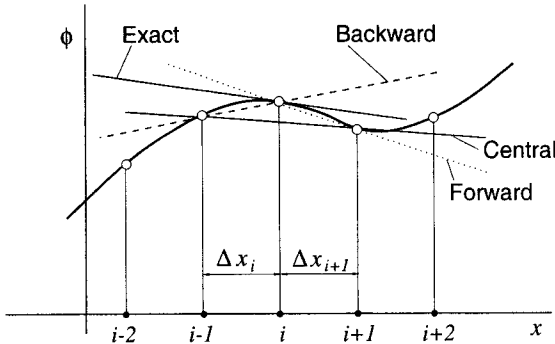
Fig. 3.2. On the definition of a derivative and its approximations

It is obvious from Fig. 3.2 that some approximations are better than others. The line for the central difference approximation has a slope very close to the slope of the exact line; if the function $\phi(x)$ were a second-order polynomial and the points were equally spaced in x-direction, the slopes would match exactly.

It is also obvious from Fig. 3.2 that the quality of the approximation improves when the additional points are close to x_i, i.e. as the grid is refined, the approximation improves. The approximations shown in Fig. 3.2 are a few of many possibilities; the following sections outline the principal approaches to deriving approximations for the first and second derivatives.

In the following two sections, only the one dimensional case is considered. The coordinate may be either Cartesian or curvilinear, the difference is of little importance here. In multidimensional finite differences, each coordinate is usually treated separately so the methods developed here are readily adapted to higher dimensionality.

3.3 Approximation of the First Derivative

Discretization of the convective term in Eq. (3.1) requires the approximation of the first derivative, $\partial(\rho u\phi)/\partial x$. We shall now describe some approaches to approximation of the first derivative of a generic variable ϕ; the methods can be applied to the first derivative of any quantity.

In the previous section, one means of deriving approximations to the first derivative was presented. There are more systematic approaches that are better suited to the derivation of more accurate approximations; some of these will be described later.

3.3.1 Taylor Series Expansion

Any continuous differentiable function $\phi(x)$ can, in the vicinity of x_i, be expressed as a Taylor series:

$$\phi(x) = \phi(x_i) + (x - x_i)\left(\frac{\partial\phi}{\partial x}\right)_i + \frac{(x-x_i)^2}{2!}\left(\frac{\partial^2\phi}{\partial x^2}\right)_i + \frac{(x-x_i)^3}{3!}\left(\frac{\partial^3\phi}{\partial x^3}\right)_i + \cdots + \frac{(x-x_i)^n}{n!}\left(\frac{\partial^n\phi}{\partial x^n}\right)_i + H \, , \tag{3.3}$$

where H means "higher order terms". By replacing x by x_{i+1} or x_{i-1} in this equation, one obtains expressions for the variable values at these points in terms of the variable and its derivatives at x_i. This can be extended to any other point near x_i, for example, x_{i+2} and x_{i-2}.

Using these expansions, one can obtain approximate expressions for the first and higher derivatives at point x_i in terms of the function values at neighboring points. For example, using Eq. (3.3) for ϕ at x_{i+1}, we can show that:

$$\left(\frac{\partial\phi}{\partial x}\right)_i = \frac{\phi_{i+1}-\phi_i}{x_{i+1}-x_i} - \frac{x_{i+1}-x_i}{2}\left(\frac{\partial^2\phi}{\partial x^2}\right)_i - \frac{(x_{i+1}-x_i)^2}{6}\left(\frac{\partial^3\phi}{\partial x^3}\right)_i + H \, . \tag{3.4}$$

Another expression may be derived using the series expression (3.3) at x_{i-1}:

$$\left(\frac{\partial \phi}{\partial x}\right)_i = \frac{\phi_i - \phi_{i-1}}{x_i - x_{i-1}} + \frac{x_i - x_{i-1}}{2}\left(\frac{\partial^2 \phi}{\partial x^2}\right)_i - \frac{(x_i - x_{i-1})^2}{6}\left(\frac{\partial^3 \phi}{\partial x^3}\right)_i + H \ . \tag{3.5}$$

Still another expression may be obtained by using Eq. (3.3) at both x_{i-1} and x_{i+1}:

$$\left(\frac{\partial \phi}{\partial x}\right)_i = \frac{\phi_{i+1} - \phi_{i-1}}{x_{i+1} - x_{i-1}} - \frac{(x_{i+1} - x_i)^2 - (x_i - x_{i-1})^2}{2\,(x_{i+1} - x_{i-1})}\left(\frac{\partial^2 \phi}{\partial x^2}\right)_i - \frac{(x_{i+1} - x_i)^3 + (x_i - x_{i-1})^3}{6\,(x_{i+1} - x_{i-1})}\left(\frac{\partial^3 \phi}{\partial x^3}\right)_i + H \ . \tag{3.6}$$

All three of these expressions are *exact* if all terms on the right hand side are retained. Because the higher-order derivatives are unknown, these expressions are not of great value as they stand. However, if the distance between the grid points i.e. $x_i - x_{i-1}$ and $x_{i+1} - x_i$ is small, the higher-order terms will be small except in the unusual situation in which the higher derivatives are locally very large. Ignoring the latter possibility, *approximations* to the first derivative result from truncating each of the series after the first terms on the right hand sides:

$$\left(\frac{\partial \phi}{\partial x}\right)_i \approx \frac{\phi_{i+1} - \phi_i}{x_{i+1} - x_i} \ ; \tag{3.7}$$

$$\left(\frac{\partial \phi}{\partial x}\right)_i \approx \frac{\phi_i - \phi_{i-1}}{x_i - x_{i-1}} \ ; \tag{3.8}$$

$$\left(\frac{\partial \phi}{\partial x}\right)_i \approx \frac{\phi_{i+1} - \phi_{i-1}}{x_{i+1} - x_{i-1}} \ . \tag{3.9}$$

These are the forward- (FDS), backward- (BDS), and central-difference (CDS) schemes mentioned earlier, respectively. The terms that were deleted from the right hand sides are called the *truncation errors;* they measure the accuracy of the approximation and determine the rate at which the error decreases as the spacing between points is reduced. In particular, the first truncated term is usually the principal source of error.

The truncation error is the sum of products of a power of the spacing between the points and a higher order derivative at the point $x = x_i$:

$$\epsilon_\tau = (\Delta x)^m \alpha_{m+1} + (\Delta x)^{m+1} \alpha_{m+2} + \cdots + (\Delta x)^n \alpha_{n+1} \ , \tag{3.10}$$

where Δx is the spacing between the points (assumed all equal for the present) and the α's are higher-order derivatives multiplied by constant factors. From Eq. (3.10) we see that the terms containing higher powers of Δx are smaller

for small spacing so that the leading term (the one with the smallest exponent) is the dominant one. As Δx is reduced, the above approximations converge to the exact derivatives with an error proportional to $(\Delta x)^m$, where m is the exponent of the leading truncation error term. The order of an approximation indicates how fast the error is *reduced* when the grid is *refined*; it does not indicate the absolute magnitude of the error. The error is thus reduced by a factor of two, four, eight or sixteen for first-, second-, third- or fourth-order approximations, respectively. It should be remembered that this rule is valid only for *sufficiently small spacings;* the definition of 'small enough' depends on the profile of the function $\phi(x)$.

3.3.2 Polynomial Fitting

An alternative way of obtaining approximations for the derivatives is to fit the function to an interpolation curve and differentiate the resulting curve. For example, if piece-wise linear interpolation is used, we obtain the FDS or BDS approximations, depending on whether the second point lies to the left or the right of point x_i.

Fitting a parabola to the data at points x_{i-1}, x_i, and x_{i+1}, and computing the first derivative at x_i from the interpolant, we obtain:

$$\left(\frac{\partial \phi}{\partial x}\right)_i = \frac{\phi_{i+1}(\Delta x_i)^2 - \phi_{i-1}(\Delta x_{i+1})^2 + \phi_i[(\Delta x_{i+1})^2 - (\Delta x_i)^2]}{\Delta x_{i+1}\Delta x_i(\Delta x_i + \Delta x_{i+1})} , \tag{3.11}$$

where $\Delta x_i = x_i - x_{i-1}$. This approximation has a second order truncation error on any grid, and is identical to the above second order approximation obtained using Taylor series approach. For uniform spacing, it reduces to the CDS approximation given above.

Other polynomials, splines etc. can be used as interpolants and then to approximate the derivative. In general, approximation of the first derivative possesses a truncation error of the same order as the degree of the polynomial used to approximate the function. We give below two third-order approximations obtained by fitting a cubic polynomial to four points and a fourth-order approximation obtained by fitting a polynomial of degree four to five points on a uniform grid:

$$\left(\frac{\partial \phi}{\partial x}\right)_i = \frac{2\,\phi_{i+1} + 3\,\phi_i - 6\,\phi_{i-1} + \phi_{i-2}}{6\,\Delta x} + \mathcal{O}\big((\Delta x)^3\big) \; ; \tag{3.12}$$

$$\left(\frac{\partial \phi}{\partial x}\right)_i = \frac{-\phi_{i+2} + 6\,\phi_{i+1} - 3\,\phi_i - 2\,\phi_{i-1}}{6\,\Delta x} + \mathcal{O}\big((\Delta x)^3\big) \; ; \tag{3.13}$$

$$\left(\frac{\partial \phi}{\partial x}\right)_i = \frac{-\phi_{i+2} + 8\,\phi_{i+1} - 8\,\phi_{i-1} + \phi_{i-2}}{12\,\Delta x} + \mathcal{O}\big((\Delta x)^4\big) \; . \tag{3.14}$$

The above approximations are third order BDS, third order FDS, and fourth order CDS schemes, respectively. On non-uniform grids, the coefficients in the above expressions become functions of grid expansion ratios.

In the case of FDS and BDS, the major contribution to the approximation comes from one side. In convection problems, BDS is sometimes used when flow is locally from node x_{i-1} to x_i and FDS when the flow is in the negative direction. Such methods are called *upwind schemes* (UDS). First order upwind schemes are very inaccurate; their truncation error has the effect of a false diffusion (i.e. the solution corresponds to a larger diffusion coefficient, which is sometimes much larger than the actual diffusivity). Higher-order upwind schemes are more accurate, but one can usually implement a CDS of higher order with less effort, since it is not necessary to check the flow direction (see above expressions).

We have demonstrated only one-dimensional polynomial fitting here; a similar approach can be used together with any type of *shape function* or interpolant in one-, two-, or three-dimensions. The only constraint is the obvious one that the number of grid points used to compute the coefficients of the shape function must equal the number of available coefficients. This approach is attractive when irregular grids are used, because it allows the possibility of avoiding the use of coordinate transformations; see Sect. 8.5.

3.3.3 Compact Schemes

For uniformly spaced grids, many special schemes can be derived. Among these are compact schemes and the spectral methods described later. Here, only Padé schemes will be described.

Compact schemes can be derived through the use of polynomial fitting. However, instead of using only the variable values at computational nodes to derive the coefficients of the polynomial, one also uses values of the derivatives at some of the points. We will use this idea to derive a fourth-order Padé scheme. The objective is to use information from near-neighbor points only; this makes solution of the resulting equations simpler and reduces the difficulty of finding approximations near the domain boundaries. In the particular schemes described here, we will use the variable values at nodes i, $i+1$, and $i-1$, and the first derivatives at nodes $i+1$ and $i-1$, to obtain an approximation for the first derivative at the node i. To this end, a polynomial of degree four is defined in the vicinity of node i:

$$\phi = a_0 + a_1(x - x_i) + a_2(x - x_i)^2 + a_3(x - x_i)^3 + a_4(x - x_i)^4 \ . \quad (3.15)$$

The coefficients $a_0, \ldots, a_4$ can be found by fitting the above polynomial to the three variable and two derivative values. However, since we are interested only in the first derivative at the node i, we only need to compute the coefficient a_1. Differentiating Eq. (3.15), we have:

$$\frac{\partial \phi}{\partial x} = a_1 + 2a_2(x - x_i) + 3a_3(x - x_i)^2 + 4a_4(x - x_i)^4 \quad (3.16)$$

so that

$$\left(\frac{\partial \phi}{\partial x}\right)_i = a_1 \ . \tag{3.17}$$

By writing Eq. (3.15) for $x = x_i$, $x = x_{i+1}$, and $x = x_{i-1}$, and Eq. (3.16) for $x = x_{i+1}$ and $x = x_{i-1}$, we obtain after some rearrangement:

$$\left(\frac{\partial \phi}{\partial x}\right)_i = -\frac{1}{4}\left(\frac{\partial \phi}{\partial x}\right)_{i+1} - \frac{1}{4}\left(\frac{\partial \phi}{\partial x}\right)_{i-1} + \frac{3}{4}\frac{\phi_{i+1} - \phi_{i-1}}{\Delta x} \ . \tag{3.18}$$

A polynomial of degree six can be used if the variable values at nodes $i+2$ and $i-2$ are added and one of degree eight can be employed if the derivatives at these two nodes are also used. An equation like Eq. (3.18) may be written at each point. The complete set of equations is actually a tridiagonal system of equations for the derivatives at the grid points. To compute the derivatives, this system has to be solved.

A family of compact centered approximations of up to sixth order can be written:

$$\alpha\left(\frac{\partial \phi}{\partial x}\right)_{i+1} + \left(\frac{\partial \phi}{\partial x}\right)_i + \alpha\left(\frac{\partial \phi}{\partial x}\right)_{i-1} = \beta\frac{\phi_{i+1} - \phi_{i-1}}{2\,\Delta x} + \gamma\frac{\phi_{i+2} - \phi_{i-2}}{4\,\Delta x} \ . \tag{3.19}$$

Depending on the choice of parameters α, β, and γ, the second- and fourth-order CDS, and fourth and sixth-order Padé schemes are obtained; the parameters and the corresponding truncation errors are listed in Table 3.1.

Table 3.1. Compact schemes: the parameters and truncation errors

Scheme	Truncation error	α	β	γ
CDS-2	$\frac{(\Delta x)^2}{3!}\frac{\partial^3\phi}{\partial x^3}$	0	1	0
CDS-4	$\frac{13(\Delta x)^4}{3\cdot 3!}\frac{\partial^5\phi}{\partial x^5}$	0	$\frac{4}{3}$	$-\frac{1}{3}$
Padé-4	$\frac{(\Delta x)^4}{5!}\frac{\partial^5\phi}{\partial x^5}$	$\frac{1}{4}$	$\frac{3}{2}$	0
Padé-6	$\frac{4(\Delta x)^6}{7!}\frac{\partial^7\phi}{\partial x^7}$	$\frac{1}{3}$	$\frac{14}{9}$	$\frac{1}{9}$

Obviously, for the same order of approximation, Padé schemes use fewer computational nodes and thus have more compact computational molecules

than central-difference approximations. If the variable values at all grids were known, we can compute the derivatives at all nodes on a grid line by solving the tridiagonal system (see Chap. 5 for details on how this can be done). We shall see, in Sect. 5.6, that these schemes can also be applied in implicit methods. This issue will be addressed again in Sect. 3.7.

The schemes derived here are only a few of the possibilities; extensions to higher order and multi-dimensional approximations are possible. It is also possible to derive schemes for non-uniform grids but the coefficients are particular to the grid, making them rather impractical.

3.3.4 Non-Uniform Grids

Since the truncation error depends not only on the grid spacing but also on the derivatives of the variable, we cannot achieve a uniform distribution of discretization error on a uniform grid. We therefore need to use a non-uniform grid. The idea is to use a smaller Δx in regions where the derivatives of the function are large and a larger Δx in regions where the function is smooth. In this way, it should be possible to spread the error nearly uniformly over the domain, thus obtaining a better solution for a given number of grid points. In this section, we will discuss the accuracy of finite difference approximations on non-uniform grids.

In some approximations, the leading term in the truncation error expression becomes zero when the spacing of the points is uniform, i.e. $x_{i+1} - x_i = x_i - x_{i-1} = \Delta x$. This is the case for the CDS approximation, see Eq. (3.6). Even though different approximations are formally of the same order for non-uniform spacing, they do not have the same truncation error. Moreover, the rate at which the error decreases when the grid is refined does not deteriorate when CDS is applied to non-uniform grids, as we shall now show.

To demonstrate this point, on which there is some confusion in the literature, note that the truncation error for the CDS is:

$$\epsilon_\tau = -\frac{(\Delta x_{i+1})^2 - (\Delta x_i)^2}{2\,(\Delta x_{i+1} + \Delta x_i)} \left(\frac{\partial^2 \phi}{\partial x^2}\right)_i - \frac{(\Delta x_{i+1})^3 + (\Delta x_i)^3}{6\,(\Delta x_{i+1} + \Delta x_i)} \left(\frac{\partial^3 \phi}{\partial x^3}\right)_i + H \,, \tag{3.20}$$

where we have used the notation (see Fig. 3.2):

$$\Delta x_{i+1} = x_{i+1} - x_i \,, \quad \Delta x_i = x_i - x_{i-1} \,.$$

The leading term is proportional to Δx, but becomes zero when $\Delta x_{i+1} = \Delta x_i$. This means that the more non-uniform the mesh spacing, the larger the error.

Let us assume that the grid expands or contracts with a constant factor r_e. This is called a compound interest grid; for it:

$$\Delta x_{i+1} = r_e \Delta x_i \ . \tag{3.21}$$

In this case, the leading truncation error term for the CDS can be rewritten:

$$\epsilon_\tau \approx \frac{(1 - r_e)\Delta x_i}{2} \left(\frac{\partial^2 \phi}{\partial x^2}\right)_i \ . \tag{3.22}$$

The leading error term of the first-order FDS or BDS schemes is:

$$\epsilon_\tau \approx \frac{\Delta x_i}{2} \left(\frac{\partial^2 \phi}{\partial x^2}\right)_i \ .$$

When r_e is close to unity, the first-order truncation error of the CDS is substantially smaller than the BDS error.

Now let us see what happens when the grid is refined. We consider two possibilities: halving the spacing between two coarse grid points and inserting new points so that the fine grid also has a constant ratio of spacings.

In the first case, the spacing is uniform around the new points, and the expansion factor r_e at the old points remains the same as on the coarse grid. If the refinement is repeated several times, we obtain a grid which is uniform everywhere except near the coarsest grid points. At this stage, at all grid points except those belonging to the coarsest grid, the spacing is uniform and the leading error term in the CDS vanishes. After some refinements, the number of points at which the spacing is non-uniform will be small. Therefore, the global error will decrease just a bit more slowly than in a true second-order scheme.
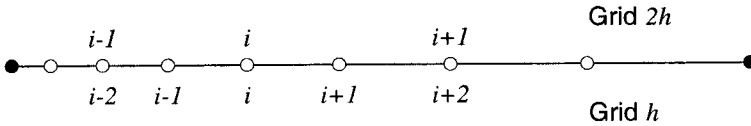
Fig. 3.3. Refinement of a non-uniform grid which expands by a constant factor r_e

In the second case the expansion factor of the fine grid is smaller than on the coarse grid. Simple arithmetic shows that

$$r_{e,h} = \sqrt{r_{e,2h}} \ , \tag{3.23}$$

where h represents the refined grid and $2h$, the coarse grid. Let us consider a node common to both grids; the ratio of the leading truncation error term at node i on the two grids is (see Eq. (3.22)):

$$r_\tau = \frac{(1 - r_e)_{2h} \, (\Delta x_i)_{2h}}{(1 - r_e)_h \, (\Delta x_i)_h} \ . \tag{3.24}$$

The following relation holds between the mesh spacing on the two grids (see Fig. 3.3):

$$(\Delta x_i)_{2h} = (\Delta x_i)_h + (\Delta x_{i-1})_h = (r_e + 1)_h (\Delta x_{i-1})_h \; .$$

When these are inserted in Eq. (3.24), taking into account Eq. (3.23), it follows that the first-order truncation error of the CDS is reduced by a factor

$$r_\tau = \frac{(1 + r_{e,h})^2}{r_{e,h}} \tag{3.25}$$

when the grid is refined. This factor has the value 4 when $r_e = 1$, i.e. when the grid is uniform. When $r_e > 1$ (expanding grid) or $r_e < 1$ (contracting grid), this factor is $r_\tau > 4$, which means that the error due to the first-order term decreases faster than the second-order error term! Since, in this method, $r_e \rightarrow 1$ as the grid is refined, the convergence becomes asymptotically second order. This will be demonstrated in the examples presented later.

A similar analysis can be performed for any scheme with the same conclusion: systematic refinement of non-uniform grids gives a rate of reduction of truncation error that has the same order as for a uniform grid.

For a given number of grid points, smaller errors are almost always obtained with non-uniform spacing. This is their purpose. However, for the grid to do its job, the user must know where smaller spacing is needed or an automatic means of grid adaptation to the solution needs to be used. An experienced user can identify regions that require fine grids; see Chap. 11 for a discussion of this issue. Methods which provide automatic error-guided grid refinement will also be presented there. It should be emphasized that grid generation becomes more difficult as the dimension of the problem is increased. Indeed, the generation of effective grids remains one of the most difficult problems in computational fluid dynamics.

Higher-order approximations of the first derivative can be obtained by using more points to eliminate more of the truncation error terms in the above expressions. For example, using ϕ_{i-1} to obtain an expression for the second derivative at x_i and substituting this expression in Eq. (3.6), we obtain the following second-order approximation (on any grid):

$$\left(\frac{\partial \phi}{\partial x}\right)_i = \frac{\phi_{i+1}(\Delta x_i)^2 - \phi_{i-1}(\Delta x_{i+1})^2 + \phi_i[(\Delta x_{i+1})^2 - (\Delta x_i)^2]}{\Delta x_{i+1}\Delta x_i(\Delta x_i + \Delta x_{i+1})} - \frac{\Delta x_{i+1}\Delta x_i}{6}\left(\frac{\partial^3 \phi}{\partial x^3}\right)_i + H \; . \tag{3.26}$$

For equispaced grids this reduces to the simple form given by Eq. (3.9).

3.4 Approximation of the Second Derivative

Second derivatives appear in the diffusive terms, see Eq. (3.1). To estimate the second derivative at a point, one may use the approximation for the first

derivative twice. This is the only approach possible when the fluid properties are variable, since we need the derivative of the product of diffusion coefficient and the first derivative. We next consider approximations to the second derivative; application to the diffusive term in the conservation equation will be discussed later.

Geometrically, the second derivative is the slope of the line tangent to the curve representing the first derivative, see Fig. 3.2. By inserting approximations for the first derivatives at locations x_{i+1} and x_i, an approximation for the second derivative is obtained:

$$\left(\frac{\partial^2 \phi}{\partial x^2}\right)_i \approx \frac{\left(\frac{\partial \phi}{\partial x}\right)_{i+1} - \left(\frac{\partial \phi}{\partial x}\right)_i}{x_{i+1} - x_i} . \tag{3.27}$$

All such approximations involve data from at least three points.

In the above equation, the outer derivative was estimated by FDS. For inner derivatives one may use a different approximation, e.g. BDS; this results in the following expression:

$$\left(\frac{\partial^2 \phi}{\partial x^2}\right)_i = \frac{\phi_{i+1}(x_i - x_{i-1}) + \phi_{i-1}(x_{i+1} - x_i) - \phi_i(x_{i+1} - x_{i-1})}{(x_{i+1} - x_i)^2 (x_i - x_{i-1})} . \tag{3.28}$$

One could also use the CDS approach which requires the first derivative at at x_{i-1} and x_{i+1}. A better choice is to evaluate $\partial \phi / \partial x$ at points halfway between x_i and x_{i+1} and x_i and x_{i-1}. The CDS approximations for these first derivatives are:

$$\left(\frac{\partial \phi}{\partial x}\right)_{i+\frac{1}{2}} \approx \frac{\phi_{i+1} - \phi_i}{x_{i+1} - x_i} \quad \text{and} \quad \left(\frac{\partial \phi}{\partial x}\right)_{i-\frac{1}{2}} \approx \frac{\phi_i - \phi_{i-1}}{x_i - x_{i-1}} ,$$

respectively. The resulting expression for the second derivative is:

$$\left(\frac{\partial^2 \phi}{\partial x^2}\right)_i \approx \frac{\left(\frac{\partial \phi}{\partial x}\right)_{i+\frac{1}{2}} - \left(\frac{\partial \phi}{\partial x}\right)_{i-\frac{1}{2}}}{\frac{1}{2}(x_{i+1} - x_{i-1})} \approx \tag{3.29}$$

$$\frac{\phi_{i+1}(x_i - x_{i-1}) + \phi_{i-1}(x_{i+1} - x_i) - \phi_i(x_{i+1} - x_{i-1})}{\frac{1}{2}(x_{i+1} - x_{i-1})(x_{i+1} - x_i)(x_i - x_{i-1})} .$$

For equidistant spacing of the points, expressions (3.28) and (3.30) become:

$$\left(\frac{\partial^2 \phi}{\partial x^2}\right)_i \approx \frac{\phi_{i+1} + \phi_{i-1} - 2\,\phi_i}{(\Delta x)^2} . \tag{3.30}$$

Taylor series expansion offers another way of deriving approximations to the second derivative. Using the series at x_{i-1} and x_{i+1} given above, one can re-derive Eq. (3.28) with an explicit expression for the error:

$$\left(\frac{\partial^2 \phi}{\partial x^2}\right)_i = \frac{\phi_{i+1}(x_i - x_{i-1}) + \phi_{i-1}(x_{i+1} - x_i) - \phi_i(x_{i+1} - x_{i-1})}{\frac{1}{2}(x_{i+1} - x_{i-1})(x_{i+1} - x_i)(x_i - x_{i-1})} -$$

$$\frac{(x_{i+1} - x_i) - (x_i - x_{i-1})}{3} \left(\frac{\partial^3 \phi}{\partial x^3}\right)_i + H \, . \tag{3.31}$$

The leading truncation error term is first order but vanishes when the spacing between the points is uniform, making the approximation second-order accurate. However, even when the grid is non-uniform, the argument given above shows that the truncation error is reduced in a second-order manner when the grid is refined. When a compound interest grid is used, the error decreases in the same way as for the CDS approximation of the first derivative, see Eq. (3.25).

Higher-order approximations for the second derivative can be obtained by including more data points, say x_{i-2} or x_{i+2}.

Finally, one can use interpolation to fit a polynomial of degree n through $n+1$ data points. From that interpolation, approximations to all derivatives up to the nth can be obtained by differentiation. Using quadratic interpolation on three points leads to the formulas given above. Approaches like those described in Sect. 3.3.3 can also be extended to the second derivative.

In general, the truncation error of the approximation to the second derivative is the degree of the interpolating polynomial minus one (first order for parabolas, second order for cubics, etc.). One order is gained when the spacing is uniform and even-order polynomials are used. For example, a polynomial of degree four fit through five points leads to a fourth-order approximation on uniform grids:

$$\left(\frac{\partial^2 \phi}{\partial x^2}\right)_i = \frac{-\phi_{i+2} + 16\,\phi_{i+1} - 30\,\phi_i + 16\,\phi_{i-1} - \phi_{i-2}}{12(\Delta x)^2} + \mathcal{O}\big((\Delta x)^4\big) \tag{3.32}$$

One can also use approximations of the second derivative to increase the accuracy of approximations to the first derivative. For example, using the FDS expression for the first derivative, Eq. (3.4), keeping just two terms on the right-hand side, and using the CDS expression (3.30) for the second derivative, results in the following expression for the first derivative:

$$\left(\frac{\partial \phi}{\partial x}\right)_i \approx \frac{\phi_{i+1}(\Delta x_i)^2 - \phi_{i-1}(\Delta x_{i+1})^2 + \phi_i[(\Delta x_{i+1})^2 - (\Delta x_i)^2]}{\Delta x_{i+1}\Delta x_i(\Delta x_i + \Delta x_{i+1})} \tag{3.33}$$

This expression possesses a second-order truncation error on any grid and reduces to the standard CDS expression for the first derivative on uniform grids. This approximation is identical to Eq. (3.26). In a similar way, one can upgrade any approximation by eliminating the derivative in the leading truncation error term. Higher-order approximations always involve more nodes, yielding more complex equations to solve and more complicated treatment

of boundary conditions so a trade-off has to be made. Second-order approximations usually offer a good combination of ease of use, accuracy, and cost-effectiveness in engineering applications. Schemes of third and fourth order offer higher accuracy for a given number of points when the grid is sufficiently fine but are more difficult to use. Methods of still higher order are used only in special cases.

For the conservative form of the diffusive term (3.1), one has to approximate the inner first derivative $\partial\phi/\partial x$ first, multiply it by Γ and differentiate the product again. As shown above, one does not have to use the same approximation for the inner and outer derivatives.

The most often used approximation is a second-order, central-difference approximation; the inner derivative is approximated at points midway between nodes, and then a central difference with a grid size Δx is used. One obtains:

$$\left[\frac{\partial}{\partial x}\left(\Gamma\frac{\partial\phi}{\partial x}\right)\right]_i \approx \frac{\left(\Gamma\frac{\partial\phi}{\partial x}\right)_{i+\frac{1}{2}} - \left(\Gamma\frac{\partial\phi}{\partial x}\right)_{i-\frac{1}{2}}}{\frac{1}{2}(x_{i+1}-x_{i-1})} \approx$$

$$\frac{\Gamma_{i+\frac{1}{2}}\frac{\phi_{i+1}-\phi_i}{x_{i+1}-x_i} - \Gamma_{i-\frac{1}{2}}\frac{\phi_i-\phi_{i-1}}{x_i-x_{i-1}}}{\frac{1}{2}(x_{i+1}-x_{i-1})} . \tag{3.34}$$

Other approximations are easily obtained using different approximations for the inner and outer first derivatives; any of the approximations presented in the previous section can be used.

3.5 Approximation of Mixed Derivatives

Mixed derivatives occur only when the transport equations are expressed in non-orthogonal coordinate systems; see Chap. 8 for an example. The mixed derivative, $\partial^2\phi/\partial x\partial y$ may be treated by combining the one-dimensional approximations as was described above for the second derivative. One can write:

$$\frac{\partial^2\phi}{\partial x\partial y} = \frac{\partial}{\partial x}\left(\frac{\partial\phi}{\partial y}\right) . \tag{3.35}$$

The mixed second derivative at (x_i, y_j) can be estimated using CDS by first evaluating the first derivative with respect to y at (x_{i+1}, y_j) and (x_{i-1}, y_j) and then evaluating the first derivative of this new function with respect to x, in the manner described above.

The order of differentiation can be changed; the numerical approximation may depend on the order. Although this may seem a drawback, it really poses no problem. All that is required is that the numerical approximation become

exact in the limit of infinitesimal grid size. The difference in the solutions obtained with two approximations is due to the discretization errors being different.

3.6 Approximation of Other Terms

In the scalar conservation equation there may be terms – which we have lumped together into the source term q_ϕ – which do not contain derivatives; these also have to be evaluated. In the FD method, only the values at the nodes are normally needed. If the non-differentiated terms involve the dependent variable, they may be expressed in terms of the nodal value of the variable. Care is needed when the dependence is non-linear. The treatment of these terms depends on the equation and further discussion is put off until Chaps. 5 and 7.

3.7 Implementation of Boundary Conditions

A finite-difference approximation to the partial differential equation is required at every interior grid point. To render the solution unique, the continuous problem requires information about the solution at the domain boundaries. Generally, the value of the variable at the boundary (Dirichlet boundary conditions) or its gradient in a particular direction (usually normal to the boundary—Neumann boundary conditions) or a linear combination of the two quantities is given.

If the variable value is known at some boundary point, then there is no need to solve for it. In all FD equations which contain data at these points, the known values are used and nothing more is necessary. A problem does arise when higher-order approximations of the derivatives are used; since they require data at more than three points, approximations at interior nodes may demand data at points beyond the boundary. It may then be necessary to use different approximations for the derivatives at points close to boundary; usually these are of lower order than the approximations used deeper in the interior and may be one-sided differences. For example, from a cubic fit to the boundary value and three inner points, Eq. (3.13) may be derived for the first derivative at the next-to-boundary point. Fitting a fourth-order polynomial through the boundary and four inner points, the following approximation for the first derivative results at $x = x_2$, the first interior point:

$$\left(\frac{\partial \phi}{\partial x}\right)_2 = \frac{-\phi_5 + 6\,\phi_4 + 18\,\phi_3 + 10\,\phi_2 - 33\,\phi_1}{60\,\Delta x} + \mathcal{O}\big((\Delta x)^4\big) \; . \qquad (3.36)$$

Approximation of the second derivative using the same polynomial gives:

$$\left(\frac{\partial^2 \phi}{\partial x^2}\right)_2 = \frac{-21\,\phi_5 + 96\,\phi_4 + 18\,\phi_3 - 240\,\phi_2 + 147\,\phi_1}{180\,(\Delta x)^2} + \mathcal{O}\big((\Delta x)^3\big)\ . \quad (3.37)$$

If the gradient is prescribed at the boundary, a suitable FD approximation for it (it must be a one-sided approximation) can be used to compute the boundary value of the variable. If, for example, zero gradient in the normal direction is prescribed, a simple FDS approximation leads to:

$$\left(\frac{\partial \phi}{\partial x}\right)_1 = 0 \quad \Rightarrow \quad \frac{\phi_2 - \phi_1}{x_2 - x_1} = 0\ , \quad (3.38)$$

which gives $\phi_1 = \phi_2$, allowing the boundary value to be replaced by the value at the node next to boundary and eliminated as an unknown. From a parabolic fit to the boundary and two inner points, the following second-order approximation, valid on any grid, is obtained for the first derivative at the boundary:

$$\left(\frac{\partial \phi}{\partial x}\right)_1 \approx \frac{-\phi_3(x_2 - x_1)^2 + \phi_2(x_3 - x_1)^2 - \phi_1[(x_3 - x_1)^2 - (x_2 - x_1)^2]}{(x_2 - x_1)(x_3 - x_1)(x_3 - x_2)}\ .$$

On a uniform grid this expression reduces to:

$$\left(\frac{\partial \phi}{\partial x}\right)_1 \approx \frac{-\phi_3 + 4\,\phi_2 - 3\,\phi_1}{2\Delta x}\ . \quad (3.39)$$

A third-order approximation on equispaced grids is obtained from a cubic fit to four points:

$$\left(\frac{\partial \phi}{\partial x}\right)_1 \approx \frac{2\,\phi_4 - 9\,\phi_3 + 18\,\phi_2 - 11\,\phi_1}{6\,\Delta x}\ . \quad (3.40)$$

Sometimes one needs to calculate first derivative normal to boundary at points at which the boundary value of the variable is given (for example, to calculate heat flux through an isothermal surface). In this case, any of the one-sided approximations given above are suitable. The accuracy of the result depends not only on the approximation used, but also on the accuracy of the values at interior points. It is sensible to use approximations of the same order for both purposes.

When the compact schemes described in Sect. 3.3.3 are used, one has to provide both the variable value and the derivative at boundary nodes. Usually, one of these is known and the other must be computed using information from the interior. For example, a one-sided approximation to the derivative at the boundary node, like Eq. (3.40), can be employed when the variable value is prescribed. On the other hand, polynomial interpolation can be used to compute the boundary value if the derivative is known. From a cubic fit to four points the following expression is obtained for the boundary value:

$$\phi_1 = \frac{18\phi_2 - 9\phi_3 + 2\phi_4}{11} - \frac{6\,\Delta x}{11}\left(\frac{\partial \phi}{\partial x}\right)_1 . \tag{3.41}$$

Approximations of lower or higher order can be obtained in a similar way.

3.8 The Algebraic Equation System

A finite-difference approximation provides an algebraic equation at each grid node; it contains the variable value at that node as well as values at neighboring nodes. If the differential equation is non-linear, the approximation will contain some non-linear terms. The numerical solution process will then require linearization; methods for solving these equations will be discussed in Chap. 5. For now, we consider only the linear case. The methods described are applicable in the non-linear case as well. For this case, the result of discretization is a system of linear algebraic equations of the form:

$$A_{\mathrm{P}}\phi_{\mathrm{P}} + \sum_l A_l\phi_l = Q_{\mathrm{P}} , \tag{3.42}$$

where P denotes the node at which the partial differential equation is approximated and index l runs over the neighbor nodes involved in finite-difference approximations. The node P and its neighbors form the so-called *computational molecule*; two examples, which result from second and third order approximations, are shown in Fig. 3.4. The coefficients A_l depend on geometrical quantities, fluid properties and, for non-linear equations, the variable values themselves. Q_{P} contains all the terms which do not contain unknown variable values; it is presumed known.
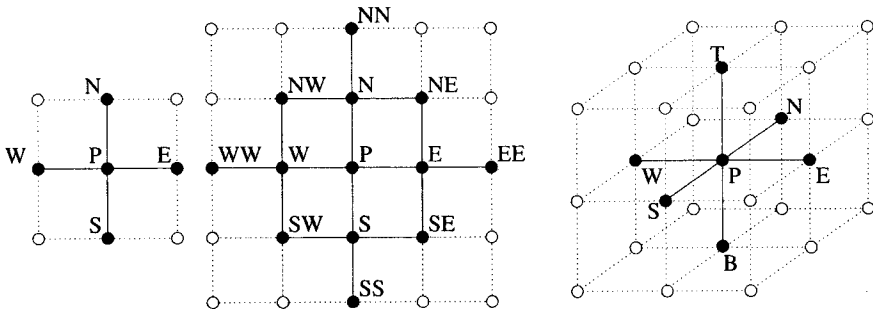
Fig. 3.4. Examples of computational molecules in 2D and 3D

The numbers of equations and unknowns must be equal, i.e., there has to be one equation for each grid node. Thus we have a large set of linear

algebraic equations, which must be solved numerically. This system is *sparse*, meaning that each equation contains only a few unknowns. The system can be written in matrix notation as follows:

$$A\phi = Q \; , \tag{3.43}$$

where A is the square sparse coefficient matrix, ϕ is a vector (or column matrix) containing the variable values at the grid nodes, and Q is the vector containing the terms on the right-hand side of Eq. (3.42).

The structure of matrix A depends on the ordering of variables in the vector ϕ. For structured grids, if the variables are labeled starting at a corner and traversing line after line in a regular manner (*lexicographic ordering*), the matrix has a poly-diagonal structure. For the case of a five-point computational molecule, all the non-zero coefficients lie on the main diagonal, the two neighboring diagonals, and two other diagonals removed by N positions from the main diagonal, where N is the number of nodes in one direction. All other coefficients are zero. This structure allows use of efficient iterative solvers.

Throughout this book we shall, for the sake of definiteness, order the entries in vector ϕ starting at the southwest corner of the domain, proceeding northwards along each grid line and then eastward across the domain (in three-dimensional cases we shall start at the bottom computational surface and proceed on each horizontal plane in the manner just described, and then go from bottom to top). The variables are normally stored in computers in one-dimensional arrays. The conversion between the grid locations, compass notation, and storage locations is indicated in Table 3.2.

Table 3.2. Conversion of grid indices to one-dimensional storage locations for vectors or column matrices

Grid location	Compass notation	Storage location
i, j, k	P	$l = (k-1)N_jN_i + (i-1)N_j + j$
$i-1, j, k$	W	$l - N_j$
$i, j-1, k$	S	$l - 1$
$i, j+1, k$	N	$l + 1$
$i+1, j, k$	E	$l + N_j$
$i, j, k-1$	B	$l - N_iN_j$
$i, j, k+1$	T	$l + N_iN_j$

Because the matrix A is sparse, it does not make sense to store it as a two-dimensional array in computer memory (this is standard practice for full matrices). Storing the elements of each non-zero diagonal in a separate array of dimension $1 \times N_iN_j$, where N_i and N_j are the numbers of grid points in the two coordinate directions, requires only $5N_iN_j$ words of storage; full array storage would require $N_i^2N_j^2$ words of storage. In three dimensions, the numbers are $7N_iN_jN_k$ and $N_i^2N_j^2N_k^2$, respectively. The difference is sufficiently

large that the diagonal-storage scheme may allow the problem to be kept in main memory when the full-array scheme does not.

If the nodal values are referenced using the grid indices, say $\phi_{i,j}$ in 2D, they look like matrix elements or components of a tensor. Since they are actually components of a vector ϕ, they should have only the single index indicated in Table 3.2.

The linearized algebraic equations in two dimensions can now be written in the form:

$$A_{l,l-N_j}\phi_{l-N_j} + A_{l,l-1}\phi_{l-1} + A_{l,l}\phi_l + \\ A_{l,l+1}\phi_{l+1} + A_{l,l+N_j}\phi_{l+N_j} = Q_l \; . \tag{3.44}$$
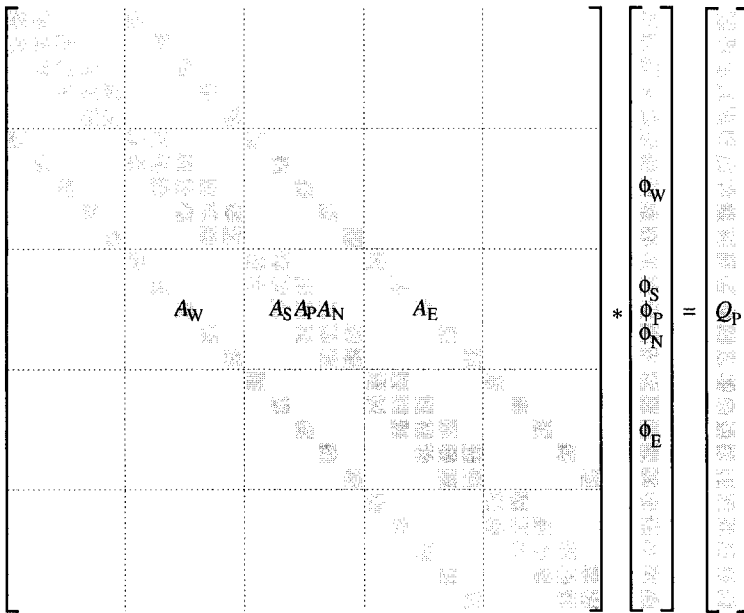
Fig. 3.5. Structure of the matrix for a five-point computational molecule (non-zero entries in the coefficient matrix on five diagonals are shaded; each horizontal set of boxes corresponds to one grid line)

As noted above, it makes little sense to store the matrix as an array. If, instead, the diagonals are kept in separate arrays, it is better to give each diagonal a separate name. Since each diagonal represents the connection to the variable at a node that lies in a particular direction with respect to the central node, we shall call them A_W, A_S, A_P, A_N and A_E; their locations in the matrix for a grid with 5×5 internal nodes are shown in Fig. 3.5. With this

ordering of points, each node is identified with an index l, which is also the relative storage location. In this notation the equation (3.44) can be written

$$A_W\phi_W + A_S\phi_S + A_P\phi_P + A_N\phi_N + A_E\phi_E = Q_P \ , \tag{3.45}$$

where the index l, which indicated rows in Eq. (3.44), is understood, and the index indicating column or location in the vector has been replaced by the corresponding letter. We shall use this shorthand notation from now on. When necessary for clarity, the index will be inserted. A similar treatment applies to three-dimensional problems.

For block-structured and composite grids, this structure is preserved within each block, and the solvers for regular structured grids may be used. This is discussed further in Chap. 5.

For unstructured grids, the coefficient matrix remains sparse, but it no longer has banded structure. For a 2D grid of quadrilaterals and approximations that use only the four nearest neighbor nodes, there are only five non-zero coefficients in any column or row. The main diagonal is full and the other non-zero coefficients lie within a certain range of the main diagonal but not necessarily on definite diagonals. A different type of iterative solver must be used for such matrices; they will be discussed in Chap. 5. The storage scheme for unstructured grids will be introduced in Chap. 8, since such grids are used mostly in complex geometries with the FV method.

3.9 Discretization Errors

Since the discretized equations represent approximations to the differential equation, the exact solution of the latter, which we shall denote by Φ, does not satisfy the difference equation. The imbalance, which is due to truncation of the Taylor series, is called *truncation error*. For a grid with a reference spacing h, the truncation error τ_h is defined as:

$$\mathcal{L}(\Phi) = L_h(\Phi) + \tau_h = 0 \ , \tag{3.46}$$

where $\mathcal{L}$ is a symbolic operator representing the differential equation and L_h is a symbolic operator representing the algebraic equation system obtained by discretization on grid h, which is given by Eq. (3.43).

The exact solution of the discretized equations on grid h, ϕ_h, satisfies the following equation:

$$L_h(\phi_h) = (A\phi - Q)_h = 0 \ . \tag{3.47}$$

It differs from the exact solution of the partial differential equation by the *discretization error*, ϵ_h^{d}, i.e.:

$$\Phi = \phi_h + \epsilon_h^{\mathrm{d}} \ . \tag{3.48}$$

From Eqs. (3.46) and (3.47) one can show that the following relation holds for linear problems:

$$L_h(\epsilon_h^{\mathrm{d}}) = -\tau_h \ . \tag{3.49}$$

This equation states that the truncation error acts as a source of the discretization error, which is convected and diffused by the operator L_h. Exact analysis is not possible for non-linear equations, but we expect similar behavior; in any case, if the error is small enough, we can locally linearize about the exact solution and what we will say in this section is valid. Information about the magnitude and distribution of the truncation error can be used as a guide for grid refinement and can help achieve the goal of having the same level of the discretization error everywhere in the solution domain. However, as the exact solution Φ is not known, the truncation error cannot be calculated exactly. An approximation to it may be obtained by using a solution from another (finer or coarser) grid. The estimate of the truncation error thus obtained is not always accurate but it serves the purpose of pointing to regions that have large errors and need finer grids.

For sufficiently fine grids, the truncation error (and the discretization error as well) is proportional to the leading term in the Taylor series:

$$\epsilon_h^{\mathrm{d}} \approx \alpha h^p + H \ , \tag{3.50}$$

where H stands for higher-order terms and α depends on the derivatives at the given point but is independent of h. The discretization error can be estimated from the difference between solutions obtained on systematically refined (or coarsened) grids. Since the exact solution may be expressed as (see Eq. (3.48)):

$$\Phi = \phi_h + \alpha h^p + H = \phi_{2h} + \alpha (2h)^p + H \ , \tag{3.51}$$

the exponent p, which is the order of the scheme, may be estimated as follows:

$$p = \frac{\log \left(\dfrac{\phi_{2h} - \phi_{4h}}{\phi_h - \phi_{2h}} \right)}{\log 2} \ . \tag{3.52}$$

From Eq. (3.51) it also follows that the discretization error on grid h can be approximated by:

$$\epsilon_h^{\mathrm{d}} \approx \frac{\phi_h - \phi_{2h}}{2^p - 1} \ . \tag{3.53}$$

If the ratio of the grid sizes on successive grids is not two, the factor 2 in the last two equations needs to be replaced by that ratio (see Roache, 1994, for details on error estimates when the grid is not systematically refined or coarsened).

When solutions on several grids are available, one can obtain an approximation of Φ which is more accurate than the solution ϕ_h on the finest grid by adding the error estimate (3.53) to ϕ_h; this method is known as *Richardson extrapolation*, (Richardson, 1910). It is simple and, when the convergence is

monotonic, accurate. When a number of solutions are available, the process can be repeated to improve the accuracy further.

We have shown above that it is the rate at which the error is reduced when the grid is refined that matters, not the formal order of the scheme as defined by the leading term in the truncation error. Equation (3.52) takes this into account and returns the correct exponent p. This estimate of the order of a scheme is also a useful tool in code validation. If a method should be, say, second-order accurate but Eq. (3.52) finds that it is only first-order accurate, there is probably an error in the code.

The order of convergence estimated using Eq. (3.52) is valid only when the convergence is monotonic. Monotonic convergence can be expected only on sufficiently fine grids. We shall show in the examples that the error dependence on grid size may be irregular when the grid is coarse. Therefore, care should be taken when comparing solutions on two grids; when convergence is not monotonic, solutions on two consecutive grids may not differ much even though the errors are not small. A third grid is necessary to assure that the solution is really converged. Also, when the solution is not smooth, the error estimates obtained with Taylor series approximations may be misleading. For example, in simulations of turbulent flows, the solution varies on a wide range of scales and the order of the solution method may not be a good indicator of solution quality. In Sect. 3.10 it will be shown that the error of a fourth-order scheme may not be much smaller than of a second-order scheme for these types of simulations.

3.10 An Introduction to Spectral Methods

Spectral methods are a class of methods less suited for general purpose CFD codes than FV and FE methods but, as they are important in some applications (e.g. simulation of turbulence), they are briefly described here. For a more complete description of them, see the book by Canuto et al. (1987).

3.10.1 Basic Concept

In spectral methods, spatial derivatives are evaluated with the aid of Fourier series or one of their generalizations. The simplest spectral method deals with periodic functions specified by their values at a uniformly spaced set of points. It is possible to represent such a function by a *discrete* Fourier series:

$$f(x_i) = \sum_{q=-N/2}^{N/2-1} \hat{f}(k_q)\, \mathrm{e}^{\mathrm{i}k_q x_i} \,, \tag{3.54}$$

where $x_i = i\,\Delta x, \quad i = 1, 2, \ldots N$ and $k_q = 2\pi q/\Delta x\, N$. Equation (3.54) can be inverted in a surprisingly simple way:

$$\hat{f}(k_q) = \frac{1}{N} \sum_{i=1}^{N} f(x_i)\, \mathrm{e}^{-\mathrm{i}k_q x_i} \,, \tag{3.55}$$

as can be proven by using the well-known formula for the summation of geometric series. The set of values of q is somewhat arbitrary; changing the index from q to $q \pm lN$, where l is an integer, produces no change in the value of $\mathrm{e}^{\pm \mathrm{i}k_q x_i}$ at the grid points. This property is known as *aliasing*; aliasing is a common and important source of error in numerical solutions of non-linear differential equations, including ones that do not use spectral methods. We shall say more about it in Chap. 9.

What makes these series useful is that Eq. (3.54) can be used to interpolate $f(x)$. We simply replace the discrete variable x_i by the continuous variable x; $f(x)$ is then defined for all x, not just the grid points. Now the choice of the range of q becomes very important. Different sets of q produce different interpolants; the best choice is the set which gives the smoothest interpolant, which is the one used in Eq. (3.54). (The set $-N/2+1, \ldots, N/2$ is as good a choice as the one selected.) Having defined the interpolant, we can differentiate it to produce a Fourier series for the derivative:

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \sum_{q=-N/2}^{N/2-1} \mathrm{i}k_q \hat{f}(k_q)\, \mathrm{e}^{\mathrm{i}k_q x} \,, \tag{3.56}$$

which shows that the Fourier coefficient of $\mathrm{d}f/\mathrm{d}x$ is $\mathrm{i}k_q \hat{f}(k_q)$. This provides a method of evaluating the derivative:

- Given $f(x_i)$, use Eq. (3.55) to compute its Fourier coefficients $\hat{f}(k_q)$;
- Compute the Fourier coefficients of $g = \mathrm{d}f/\mathrm{d}x$; $\hat{g}(k_q) = \mathrm{i}k_q \hat{f}(k_q)$;
- Evaluate the series (3.56) to obtain $g = \mathrm{d}f/\mathrm{d}x$ at the grid points.

Several points need to be noted.

- The method is easily generalized to higher derivatives; for example, the Fourier coefficient of $\mathrm{d}^2 f/\mathrm{d}x^2$ is $-k_q^2 \hat{f}(k_q)$.
- The error in the computed derivative decreases exponentially with N when the number of grid points N is large if $f(x)$ is periodic in x. This makes spectral methods much more accurate than finite difference methods for large N; however, for small N, this may not be the case. The definition of 'large' depends on the function.
- The cost of computing the Fourier coefficients using Eq. (3.55) and/or the inverse using Eq. (3.54), if done in the most obvious manner, scales as N^2. This would be prohibitively expensive; the method is made practical by the existence of a fast method of computing Fourier transform (FFT) for which the cost is proportional to $N \log_2 N$.

To obtain the advantages of this particular spectral method, the function must be periodic and the grid points uniformly spaced. These conditions

can be relaxed by using functions other than complex exponentials but any change in geometry or boundary conditions requires a considerable change in the method, making spectral methods relatively inflexible. For the problems to which they are ideally suited (for example, the simulation of turbulence in geometrically simple domains), they are unsurpassed.

3.10.2 Another View of Discretization Error

Spectral methods are as useful for providing another way of looking at truncation errors as they are as computational methods on their own. So long as we deal with periodic functions, the series (3.54) represents the function and we may approximate its derivative by any method we choose. In particular, we can use the exact spectral method of the example above or a finite difference approximation. Any of these methods can be applied term-by-term to the series so it is sufficient to consider differentiation of $\mathrm{e}^{\mathrm{i}kx}$. The exact result is $\mathrm{i}k\mathrm{e}^{\mathrm{i}kx}$. On the other hand, if we apply the central difference operator of Eq. (3.9) to this function we find:

$$\frac{\delta \mathrm{e}^{\mathrm{i}kx}}{\delta x} = \frac{\mathrm{e}^{\mathrm{i}k(x+\Delta x)} - \mathrm{e}^{\mathrm{i}k(x-\Delta x)}}{2\Delta x} = \mathrm{i}\,\frac{\sin(k\,\Delta x)}{\Delta x}\,\mathrm{e}^{\mathrm{i}kx} = \mathrm{i}k_{\mathrm{eff}}\,\mathrm{e}^{\mathrm{i}kx} \,, \tag{3.57}$$

where k_{eff} is called the *effective wavenumber* because using the finite difference approximation is equivalent to replacing the exact wavenumber k by k_{eff}. Similar expressions can be derived for other schemes; for example, the fourth order CDS, Eq. (3.14), leads to:

$$k_{\mathrm{eff}} = \frac{\sin(k\,\Delta x)}{3\Delta x}\,[4 - \cos(k\,\Delta x)] \,. \tag{3.58}$$

For low wavenumber (corresponding to smooth functions), the effective wavenumber of the CDS approximation can be expanded in a Taylor series:

$$k_{\mathrm{eff}} = \frac{\sin(k\,\Delta x)}{\Delta x} = k - \frac{k^3(\Delta x)^2}{6} \,, \tag{3.59}$$

which shows the second-order nature of the approximation for small k and small Δx. However, in any computation, wavenumbers up to $k_{\mathrm{max}} = \pi/\Delta x$ may be encountered. The magnitude of a given Fourier coefficient depends on the function whose derivatives are being approximated; smooth functions have small high wavenumber components but rapidly varying functions give Fourier coefficients that decrease slowly with wavenumber.

In Fig. 3.6 the effective wavenumbers of the second and fourth order CDS scheme, normalized by k_{max}, are shown as functions of the normalized wavenumber $k^* = k/k_{\mathrm{max}}$. Both schemes give a poor approximation if the wavenumber is larger than half the maximum value. More wavenumbers are included as the grid is refined. In the limit of small spacings, the function is
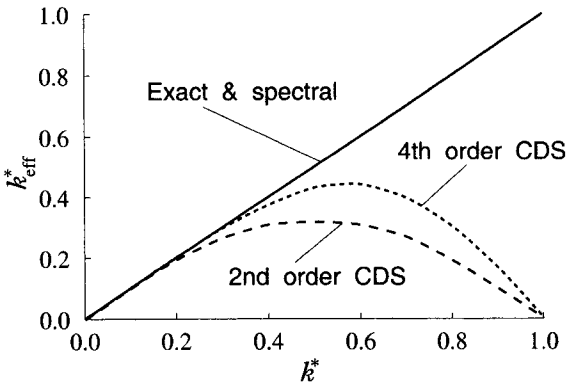
Fig. 3.6. Effective wavenumber for the second and fourth order CDS approximation of the first derivative, normalized by $k_{\text{max}} = \pi/\Delta x$

smooth relative to the grid, only the small wavenumbers have large coefficients, and accurate results may be expected.

If we are solving a problem with a solution that is not very smooth, the order of the discretization method may no longer be a good indicator of its accuracy. One needs to be very careful about claims that a particular scheme is accurate because the method used is of high order. The result is accurate only if there are enough nodes per wavelength of a highest wavenumber in the solution.

Spectral methods yield an error that decreases more rapidly than any power of the grid size as the latter goes to zero. This is often cited as an advantage of the method. However, this behavior is obtained only when enough points are used (the definition of 'enough' depends on the function). For small numbers of grid points, spectral methods may actually yield larger errors than finite difference methods.

Finally, we note that the effective wavenumber of the upwind difference method is:

$$k_{\text{eff}} = \frac{1 - \mathrm{e}^{-\mathrm{i}k\,\Delta x}}{\Delta x} \tag{3.60}$$

and is complex. This is an indication of the dissipative nature of this approximation.

3.11 Example

In this example we solve the steady 1D convection/diffusion equation with Dirichlet boundary conditions at both ends. The aim is to demonstrate the properties of the FD discretization technique for a simple problem which has an analytic solution.

The equation to be solved reads (see Eq. (1.28)):
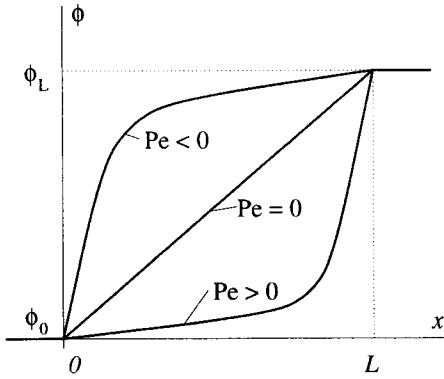
Fig. 3.7. Boundary conditions and solution profiles for the 1D problem as a function of the Peclet number

$$\frac{\partial(\rho u \phi)}{\partial x} = \frac{\partial}{\partial x}\left(\Gamma \frac{\partial \phi}{\partial x}\right) , \tag{3.61}$$

with the boundary conditions: $\phi = \phi_0$ at $x = 0$, $\phi = \phi_L$ at $x = L$, see Fig. 3.7; the partial derivatives may be replaced by ordinary derivatives in this case. The density ρ and the velocity u are assumed constant. This problem has the exact solution:

$$\phi = \phi_0 + \frac{\mathrm{e}^{x\mathrm{Pe}/L} - 1}{\mathrm{e}^{\mathrm{Pe}} - 1}\left(\phi_L - \phi_0\right) . \tag{3.62}$$

Here Pe is the Peclet number, defined as:

$$\mathrm{Pe} = \frac{\rho u L}{\Gamma} . \tag{3.63}$$

Because it is so simple, this problem is often used as a test of numerical methods, including both discretization and solution schemes. Physically, it represents a situation in which convection is balanced by diffusion in the streamwise direction. There are few actual flows in which this balance plays an important role. Normally, convection is balanced by either a pressure gradient or diffusion in the direction normal to the flow. In the literature, one finds many methods that were developed for Eq. (3.61) and then applied to the Navier-Stokes equations. The results are often very poor and most of these methods are best avoided. Indeed, use of this problem as a test case has probably produced more poor choices of method than any other in the field. Despite these difficulties, we shall consider this problem as some of the issues it raises are worthy of attention.

Let us consider the case $u \geq 0$ and $\phi_0 < \phi_L$; other situations are easily dealt with. In the case of small velocity ($u \approx 0$) or large diffusivity Γ, the Peclet number tends to zero and convection can be neglected; the solution is then linear in x. When the Peclet number is large, ϕ grows slowly with x and then suddenly rises to ϕ_L over a short distance close to $x = L$. The

sudden change in the gradient of ϕ provides a severe test of the discretization method.

We shall discretize Eq. (3.61) using FD schemes which use the three-point computational molecule. The resulting algebraic equation at node i reads:

$$A_{\rm P}^{i}\phi_i + A_{\rm E}^{i}\phi_{i+1} + A_{\rm W}^{i}\phi_{i-1} = Q_i \; . \tag{3.64}$$

It is common practice to discretize the diffusion term using CDS; thus, for the outer derivative, we have:

$$-\left[\frac{\partial}{\partial x}\left(\Gamma\frac{\partial\phi}{\partial x}\right)\right]_i \approx -\frac{\left(\Gamma\dfrac{\partial\phi}{\partial x}\right)_{i+\frac{1}{2}} - \left(\Gamma\dfrac{\partial\phi}{\partial x}\right)_{i-\frac{1}{2}}}{\frac{1}{2}(x_{i+1}-x_{i-1})} \; . \tag{3.65}$$

The CDS approximations of the inner derivatives are:

$$\left(\Gamma\frac{\partial\phi}{\partial x}\right)_{i+\frac{1}{2}} \approx \Gamma\,\frac{\phi_{i+1}-\phi_i}{x_{i+1}-x_i} \; ; \quad \left(\Gamma\frac{\partial\phi}{\partial x}\right)_{i-\frac{1}{2}} \approx \Gamma\,\frac{\phi_i-\phi_{i-1}}{x_i-x_{i-1}} \; . \tag{3.66}$$

The contributions of the diffusion term to the coefficients of the algebraic equation (3.64) are thus:

$$A_{\rm E}^{\rm d} = -\frac{2\,\Gamma}{(x_{i+1}-x_{i-1})(x_{i+1}-x_i)} \; ;$$

$$A_{\rm W}^{\rm d} = -\frac{2\,\Gamma}{(x_{i+1}-x_{i-1})(x_i-x_{i-1})} \; ;$$

$$A_{\rm P}^{\rm d} = -(A_{\rm E}^{\rm d} + A_{\rm W}^{\rm d}) \; .$$

If the convection term is discretized using first-order upwind differences (UDS – FDS or BDS, depending on the flow direction), we have:

$$\left[\frac{\partial(\rho u\phi)}{\partial x}\right]_i \approx \begin{cases} \rho u\dfrac{\phi_i-\phi_{i-1}}{x_i-x_{i-1}} \; , \text{ if } u>0 \; ; \\ \rho u\dfrac{\phi_{i+1}-\phi_i}{x_{i+1}-x_i} \; , \text{ if } u<0 \; . \end{cases} \tag{3.67}$$

This leads to the following contributions to the coefficients of Eq. (3.64):

$$A_{\rm E}^{\rm c} = \frac{\min(\rho u,0)}{x_{i+1}-x_i} \; ; \qquad A_{\rm W}^{\rm c} = -\frac{\max(\rho u,0)}{x_i-x_{i-1}} \; ;$$

$$A_{\rm P}^{\rm c} = -(A_{\rm E}^{\rm c} + A_{\rm W}^{\rm c}) \; .$$

Either $A_{\rm E}^{\rm c}$ or $A_{\rm W}^{\rm c}$ is zero, depending on the flow direction.

The CDS approximation leads to:

$$\left[\frac{\partial(\rho u \phi)}{\partial x}\right]_i \approx \rho u \frac{\phi_{i+1} - \phi_{i-1}}{x_{i+1} - x_{i-1}} \; . \tag{3.68}$$

The CDS contributions to the coefficients of Eq. (3.64) are:

$$A_{\mathrm{E}}^{\mathrm{c}} = \frac{\rho u}{x_{i+1} - x_{i-1}} \; ; \qquad A_{\mathrm{W}}^{\mathrm{c}} = -\frac{\rho u}{x_{i+1} - x_{i-1}} \; ;$$

$$A_{\mathrm{P}}^{\mathrm{c}} = -(A_{\mathrm{E}}^{\mathrm{c}} + A_{\mathrm{W}}^{\mathrm{c}}) = 0 \; .$$

The total coefficients are the sums of the convection and diffusion contributions, A^{c} and A^{d}.

The values of ϕ at boundary nodes are specified: $\phi_1 = \phi_0$ and $\phi_N = \phi_L$, where N is the number of nodes including the two at the boundaries. This means that, for the node at $i = 2$, the term $A_{\mathrm{W}}^2 \phi_1$ can be calculated and added to Q_2, the right hand side, and we set the coefficient A_{W}^2 in that equation to zero. Analogously, we add the product $A_{\mathrm{E}}^{N-1} \phi_N$ for the node $i = N - 1$ to Q_{N-1} and set the coefficient $A_{\mathrm{E}}^{N-1} = 0$.

The resulting tridiagonal system is easily solved. We shall only discuss the solutions here; the solver used to obtain them will be introduced in Chap. 5.
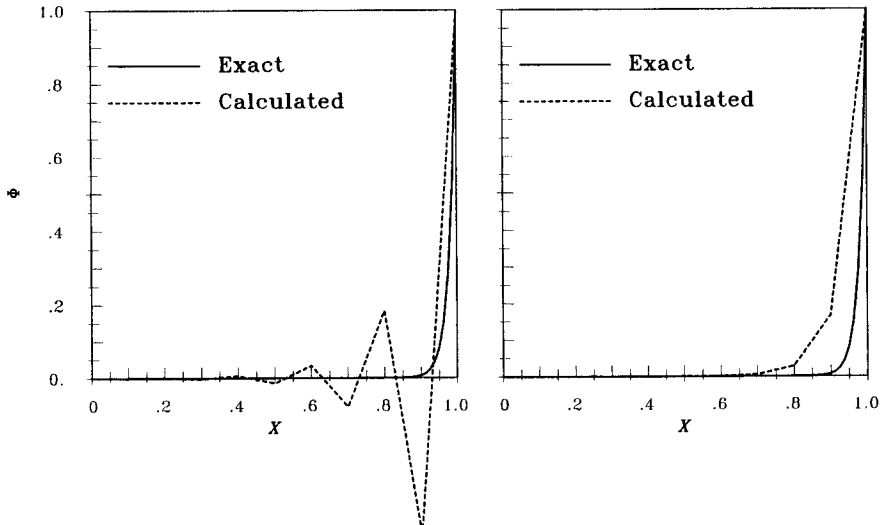
Fig. 3.8. Solution of the 1D convection/diffusion equation at Pe = 50 using CDS (left) and UDS (right) for convection terms and a uniform grid with 11 nodes

In order to demonstrate the false diffusion associated with UDS and the possibility of oscillations when using CDS, we shall consider the case with Pe = 50 ($L = 1.0$, $\rho = 1.0$, $u = 1.0$, $\Gamma = 0.02$, $\phi_0 = 0$ and $\phi_L = 1.0$). We

start with results obtained using uniform grid with 11 nodes (10 equal subdivisions). The profiles of $\phi(x)$ obtained using CDS and UDS for convection and CDS for diffusion terms are shown in Fig. 3.8.

The UDS solution is obviously over-diffusive; it corresponds to the exact solution for Pe $\approx$ 18 (instead of 50). The false diffusion is stronger than the true diffusion! On the other hand, the CDS solution exhibits severe oscillations. The oscillations are due to the sudden change of gradient in ϕ at the last two points. The Peclet number based on mesh spacing (see Eq. (3.63)) is equal to 5 at every node.
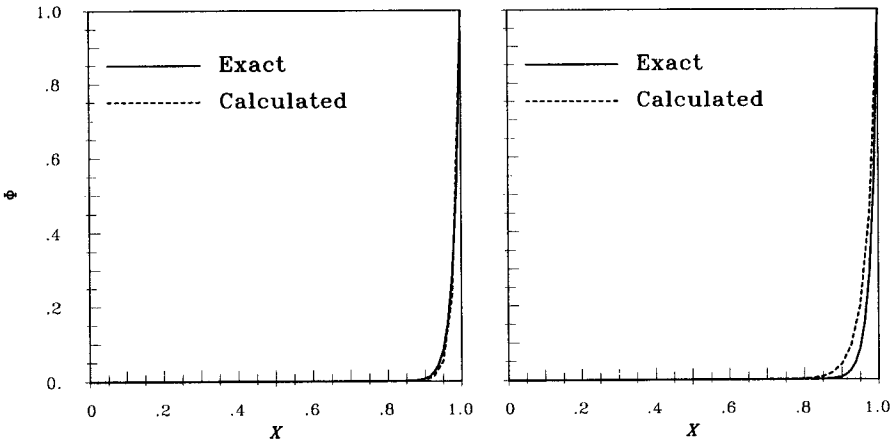
Fig. 3.9. Solution of the 1D convection/diffusion equation at Pe = 50 using CDS (left) and UDS (right) for convection terms and a uniform grid with 41 nodes

If the grid is refined, the CDS oscillations are reduced, but they are still present when 21 points are used. After the second refinement (41 grid nodes), the solution is oscillation-free and very accurate, see Fig. 3.9. The accuracy of the UDS solution has also been improved by grid refinement, but it is still substantially in error for $x > 0.8$.

The CDS oscillations depend on the value of the local Peclet number, $\mathrm{Pe} = \rho u \Delta x / \Gamma$. It can be shown that no oscillations occur if the local Peclet number is $\mathrm{Pe} \leq 2$ at every grid node (see Patankar, 1980). This is a sufficient, but not necessary condition for boundedness of CDS solution. The so-called *hybrid scheme* (Spalding, 1972) was designed to switch from CDS to UDS at any node at which $\mathrm{Pe} \geq 2$. This is too restrictive and reduces the accuracy. Oscillations appear only when the solution changes rapidly in a region of high local Peclet number.

In order to demonstrate this, we repeat the calculation using a non-uniform grid with 11 nodes. The smallest and the largest mesh spacings are $\Delta x_{\min} = x_N - x_{N-1} = 0.0125$ and $\Delta x_{\max} = x_2 - x_1 = 0.31$, corresponding to
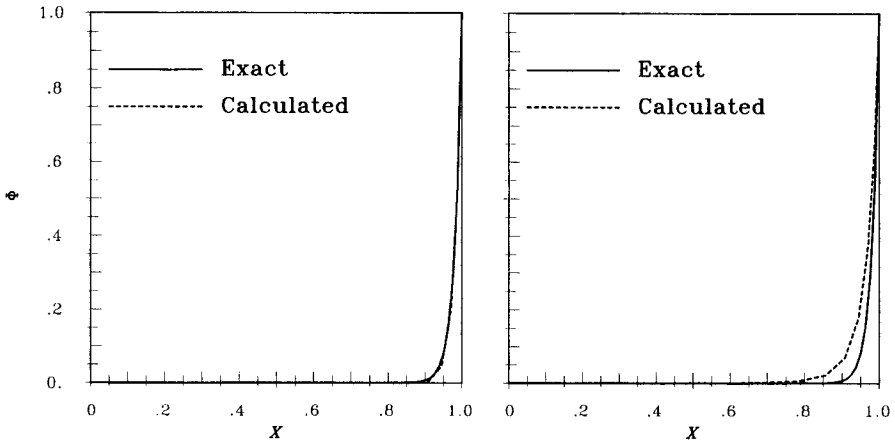
Fig. 3.10. Solution of the 1D convection/diffusion equation at Pe = 50 using CDS (left) and UDS (right) for convection terms and a non-uniform grid with 11 nodes (grid dense near right end)

an expansion factor $r_e = 0.7$, see Eq. (3.21). The minimum Peclet number is thus $\mathrm{Pe}_{\mathrm{min}} = 0.625$ near right boundary, and the maximum is $\mathrm{Pe}_{\mathrm{max}} = 15.5$ near left boundary. The Peclet number is thus smaller than two in the region where ϕ undergoes strong change and is large in the region of nearly constant ϕ. The calculated profiles on this grid using the CDS and UDS schemes are shown in Fig. 3.10. There are no oscillations in the CDS solution. Moreover, it is as accurate as the solution on a uniform grid with four times as many nodes. The accuracy of the UDS solution has also been improved by using a non-uniform grid but is still unacceptable.
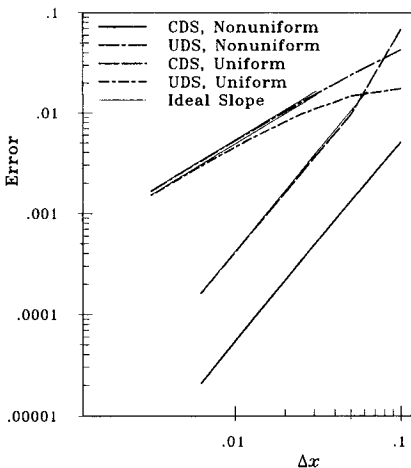
Fig. 3.11. Average error in the solution of the 1D convection/diffusion equation for Pe=50 as a function of the average mesh spacing

Since this problem has an analytic solution, Eq. (3.62), we can calculate the error in the numerical solution directly. The following average error is used as a measure:

$$\epsilon = \frac{\sum_i |\phi_i^{\text{exact}} - \phi_i|}{N} .$$

The problem was solved using both CDS and UDS and both uniform and non-uniform grids with up to 321 nodes. The average error is plotted as a function of average mesh spacing in Fig. 3.11. The UDS error asymptotically approaches the slope expected of a first-order scheme. The CDS shows, from the second grid onwards, the slope expected of a second-order scheme: the error is reduced by two orders of magnitude when the grid spacing is reduced one order of magnitude.

This example clearly shows that the solution on a non-uniform grid converges in the same way as the solution on a uniform grid, even though the truncation error contains a first-order term as explained in Sect. 3.3.4. For the CDS, the average error on a non-uniform grid is almost an order of magnitude smaller than on a uniform grid with the same number of grid nodes. This is due to the fact that the mesh spacing is small where the error would be large. That Fig. 3.11 indicates larger error for UDS on a non-uniform than on a uniform grid is due to the fact that large errors at a few nodes on a uniform grid have a small effect on the average; maximum nodal error is much larger on uniform than on non-uniform grids, as can be seen by examining Figs. 3.8 and 3.10.

For related examples, see the last section of the next chapter.

4. Finite Volume Methods

4.1 Introduction

As in the previous chapter, we consider only the generic conservation equation for a quantity ϕ and assume that the velocity field and all fluid properties are known. The finite volume method uses the integral form of the conservation equation as the starting point:

$$\int_S \rho\phi \boldsymbol{v} \cdot \boldsymbol{n}\, \mathrm{d}S = \int_S \Gamma \operatorname{grad}\phi \cdot \boldsymbol{n}\, \mathrm{d}S + \int_\Omega q_\phi\, \mathrm{d}\Omega \,. \tag{4.1}$$

The solution domain is subdivided into a finite number of small control volumes (CVs) by a grid which, in contrast to the finite difference (FD) method, defines the control volume boundaries, not the computational nodes. For the sake of simplicity we shall demonstrate the method using Cartesian grids; complex geometries are treated in Chap. 8.

The usual approach is to define CVs by a suitable grid and assign the computational node to the CV center. However, one could as well (for structured grids) define the nodal locations first and construct CVs around them, so that CV faces lie midway between nodes; see Fig. 4.1. Nodes on which boundary conditions are applied are shown as full symbols in this figure.
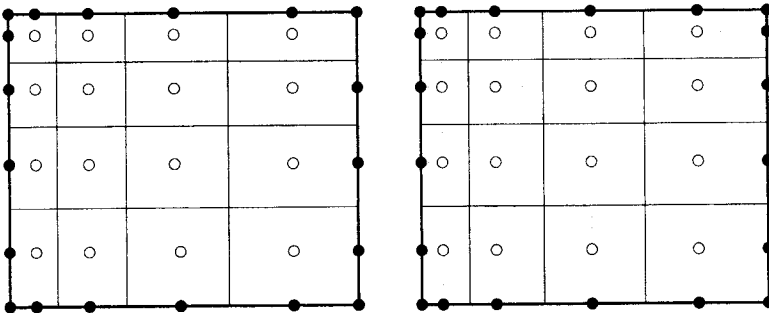
Fig. 4.1. Types of FV grids: nodes centered in CVs (left) and CV faces centered between nodes (right)

The advantage of the first approach is that the nodal value represents the mean over the CV volume to higher accuracy (second order) than in the second approach, since the node is located at the centroid of the CV. The advantage of the second approach is that CDS approximations of derivatives at CV faces are more accurate when the face is midway between two nodes. The first variant is used more often and will be adopted in this book.

There are several other specialized variants of FV-type methods (cell-vertex schemes, dual-grid schemes etc.); some of these will be described later in this chapter and in Chap. 8. Here we shall describe just the basic method. The discretization principles are the same for all variants – one only has to take into account the relation between the various locations within the integration volume.

The integral conservation equation (4.1) applies to each CV, as well as to the solution domain as a whole. If we sum equations for all CVs, we obtain the global conservation equation, since surface integrals over inner CV faces cancel out. Thus global conservation is built into the method and this provides one of its principal advantages.

To obtain an algebraic equation for a particular CV, the surface and volume integrals need be approximated using quadrature formulae. Depending on the approximations used, the resulting equations may or may not be those obtained from the FD method.

4.2 Approximation of Surface Integrals

In Figs. 4.2 and 4.3, typical 2D and 3D Cartesian control volumes are shown together with the notation we shall use. The CV surface consists of four (in 2D) or six (in 3D) plane faces, denoted by lower-case letters corresponding to their direction (e, w, n, s, t, and b) with respect to the central node (P). The 2D case can be regarded as a special case of the 3D one in which the dependent variables are independent of z. In this chapter we shall deal mostly with 2D grids; the extension to 3D problems is straightforward.

The net flux through the CV boundary is the sum of integrals over the four (in 2D) or six (in 3D) CV faces:

$$\int_S f \, \mathrm{d}S = \sum_k \int_{S_k} f \, \mathrm{d}S \,, \tag{4.2}$$

where f is the component of the convective ($\rho\phi\boldsymbol{v}\cdot\boldsymbol{n}$) or diffusive ($\Gamma \operatorname{grad}\phi\cdot\boldsymbol{n}$) flux vector in the direction normal to CV face. As the velocity field and the fluid properties are assumed known, the only unknown is ϕ. If the velocity field is not known, we have a more complex problem involving non-linear coupled equations; we shall deal with it in Chap. 7.

For maintenance of conservation, it is important that CVs do not overlap; each CV face is unique to the two CVs which lie on either side of it.
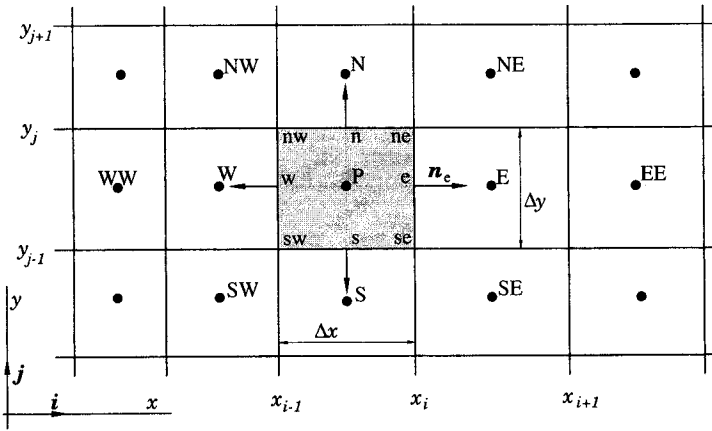
Fig. 4.2. A typical CV and the notation used for a Cartesian 2D grid
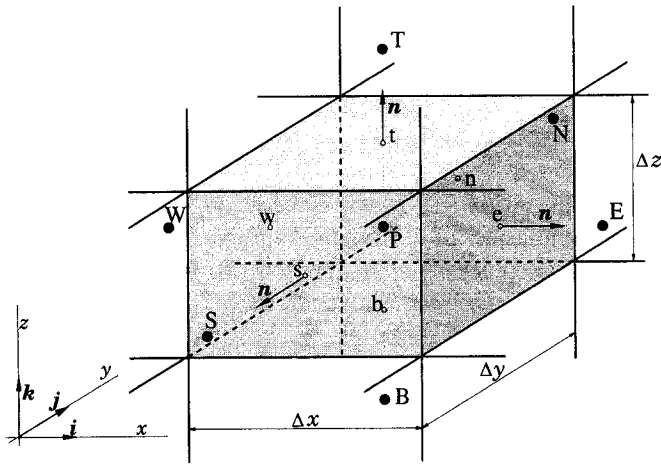
Fig. 4.3. A typical CV and the notation used for a Cartesian 3D grid

In what follows, only a typical CV face, the one labeled 'e' in Fig. 4.2 will be considered; analogous expressions may be derived for all faces by making appropriate index substitutions.

To calculate the surface integral in Eq. (4.2) exactly, one would need to know the integrand f everywhere on the surface S_e. This information is not available, as only the nodal (CV center) values of ϕ are calculated so an approximation must be introduced. This is best done using two levels of approximation:

- the integral is approximated in terms of the variable values at one or more locations on the cell face;

- the cell-face values are approximated in terms of the nodal (CV center) values.

The simplest approximation to the integral is the midpoint rule: the integral is approximated as a product of the integrand at the cell-face center (which is itself an approximation to the mean value over the surface) and the cell-face area:

$$F_{\mathrm{e}} = \int_{S_{\mathrm{e}}} f \, \mathrm{d}S = \overline{f}_{\mathrm{e}} S_{\mathrm{e}} \approx f_{\mathrm{e}} S_{\mathrm{e}} \ . \tag{4.3}$$

This approximation of the integral – provided the value of f at location 'e' is known – is of second-order accuracy.

Since the value of f is not available at the cell face center 'e', it has to be obtained by interpolation. In order to preserve the second-order accuracy of the midpoint rule approximation of the surface integral, the value of f_{e} has to be computed with at least second-order accuracy. We shall present some widely used approximations in Sect. 4.4.

Another second-order approximation of the surface integral in 2D is the trapezoid rule, which leads to:

$$F_{\mathrm{e}} = \int_{S_{\mathrm{e}}} f \, \mathrm{d}S \approx \frac{S_{\mathrm{e}}}{2} \left(f_{\mathrm{ne}} + f_{\mathrm{se}} \right) \ . \tag{4.4}$$

In this case we need to evaluate the flux at the CV corners.

For higher-order approximation of the surface integrals, the flux must be evaluated at more than two locations. A fourth-order approximation is Simpson's rule, which estimates the integral over S_{e} as:

$$F_{\mathrm{e}} = \int_{S_{\mathrm{e}}} f \, \mathrm{d}S \approx \frac{S_{\mathrm{e}}}{6} \left(f_{\mathrm{ne}} + 4 f_{\mathrm{e}} + f_{\mathrm{se}} \right) \ . \tag{4.5}$$

Here the values of f are needed at three locations: the cell face center 'e' and the two corners, 'ne' and 'se'. In order to retain the fourth-order accuracy these values have to be obtained by interpolation of the nodal values at least as accurate as Simpson's rule. Cubic polynomials are suitable, as shown below.

In 3D, the midpoint rule is again the simplest second-order approximation. Higher-order approximations, which require the integrand at locations other than cell face center (e.g. corners and centers of edges) are possible, but they are more difficult to implement. One possibility is mentioned in the following section.

If the variation of f is assumed to have some particular simple shape (e.g. an interpolation polynomial), the integration is easy. The accuracy of the approximation then depends on the order of shape functions.

4.3 Approximation of Volume Integrals

Some terms in the transport equations require integration over the volume of a CV. The simplest second-order accurate approximation is to replace the volume integral by the product of the mean value of the integrand and the CV volume and approximate the former as the value at the CV center:

$$Q_{\mathrm{P}} = \int_{\Omega} q \,\mathrm{d}\Omega = \overline{q}\, \Delta\Omega \approx q_{\mathrm{P}}\, \Delta\Omega \ , \tag{4.6}$$

where q_{P} stands for the value of q at the CV center. This quantity is easily calculated; since all variables are available at node P, no interpolation is necessary. The above approximation becomes exact if q is either constant or varies linearly within the CV; otherwise, it contains a second-order error, as is easily shown.

An approximation of higher order requires the values of q at more locations than just the center. These values have to be obtained by interpolating nodal values or, equivalently, by using shape functions.

In 2D the volume integral becomes an area integral. A fourth-order approximation can be obtained by using the bi-quadratic shape function:

$$\begin{aligned} q(x,y) = {} & a_0 + a_1\, x + a_2\, y + a_3\, x^2 + a_4\, y^2 + \\ & a_5\, xy + a_6\, x^2 y + a_7\, xy^2 + a_8\, x^2 y^2 \ . \end{aligned} \tag{4.7}$$

The nine coefficients are obtained by fitting the function to the values of q at nine locations ('nw', 'w', 'sw', 'n', P, 's', 'ne', 'e' and 'se', see Fig. 4.2). The integral can then be evaluated. In 2D the integration gives (for Cartesian grids):

$$\begin{aligned} Q_{\mathrm{P}} = \int_{\Omega} q \,\mathrm{d}\Omega \approx \Delta x\, \Delta y \Big[& a_0 + \frac{a_3}{12}\, (\Delta x)^2 + \\ & \frac{a_4}{12}\, (\Delta y)^2 + \frac{a_8}{144}\, (\Delta x)^2 (\Delta y)^2 \Big] \ . \end{aligned} \tag{4.8}$$

Only four coefficients need to be determined, but they depend on the values of q at all nine locations listed above. On a uniform Cartesian grid we obtain:

$$\begin{aligned} Q_{\mathrm{P}} = \frac{\Delta x\, \Delta y}{36} \big(& 16\, q_{\mathrm{P}} + 4\, q_{\mathrm{s}} + 4\, q_{\mathrm{n}} + 4\, q_{\mathrm{w}} + 4\, q_{\mathrm{e}} + \\ & q_{\mathrm{se}} + q_{\mathrm{sw}} + q_{\mathrm{ne}} + q_{\mathrm{nw}} \big) \ . \end{aligned} \tag{4.9}$$

Since only the value at P is available, interpolation has to be used to obtain q at the other locations. It has to be at least fourth-order accurate to retain the accuracy of the integral approximation. Some possibilities will be described in the next section.

The above fourth-order approximation of the volume integral in 2D can be used to approximate the surface integrals in 3D. Higher-order approximations of volume integrals in 3D are more complex, but can be found using the same techniques.

4.4 Interpolation and Differentiation Practices

The approximations to the integrals require the values of variables at locations other than computational nodes (CV centers). The integrand, denoted in the previous sections by f, involves the product of several variables and/or variable gradients at those locations: $f^{\mathrm{c}} = \rho\phi\boldsymbol{v}\cdot\boldsymbol{n}$ for the convective flux and $f^{\mathrm{d}} = \Gamma \operatorname{grad}\phi \cdot \boldsymbol{n}$ for the diffusive flux. We assume that the velocity field and the fluid properties ρ and Γ are known at all locations. To calculate the convective and diffusive fluxes, the value of ϕ and its gradient normal to the cell face at one or more locations on the CV surface are needed. Volume integrals of the source terms may also require these values. They have to be expressed in terms of the nodal values by interpolation. Numerous possibilities are available; we shall mention a few that are most commonly used. In particular we shall show how the value of ϕ and its normal derivative at cell face 'e' can be approximated.

4.4.1 Upwind Interpolation (UDS)

Approximating ϕ_{e} by its value at the node upstream of 'e' is equivalent to using a backward- or forward-difference approximation for the first derivative (depending on the flow direction), hence the name *upwind differencing scheme* (UDS) for this approximation. In UDS, ϕ_{e} is approximated as:

$$\phi_{\mathrm{e}} = \begin{cases} \phi_{\mathrm{P}} \text{ if } (\boldsymbol{v}\cdot\boldsymbol{n})_{\mathrm{e}} > 0 \,; \\ \phi_{\mathrm{E}} \text{ if } (\boldsymbol{v}\cdot\boldsymbol{n})_{\mathrm{e}} < 0 \,. \end{cases} \tag{4.10}$$

This is the only approximation that unconditionally satisfies the boundedness criterion i.e. it will never yield oscillatory solutions. However, it achieves this by being *numerically diffusive.* This was shown in the preceding chapter and will be shown again below.

Taylor series expansion about P gives (for Cartesian grid and $(\boldsymbol{v}\cdot\boldsymbol{n})_{\mathrm{e}} > 0$):

$$\phi_{\mathrm{e}} = \phi_{\mathrm{P}} + (x_{\mathrm{e}} - x_{\mathrm{P}}) \left(\frac{\partial \phi}{\partial x}\right)_{\mathrm{P}} + \frac{(x_{\mathrm{e}} - x_{\mathrm{P}})^2}{2} \left(\frac{\partial^2 \phi}{\partial x^2}\right)_{\mathrm{P}} + H \,, \tag{4.11}$$

where H denotes higher-order terms. The UDS approximation retains only the first term on the right-hand side, so it is a first-order scheme. Its leading truncation error term is diffusive i.e. it resembles a diffusive flux:

$$f_{\mathrm{e}}^{\mathrm{d}} = \Gamma_{\mathrm{e}} \left(\frac{\partial \phi}{\partial x}\right)_{\mathrm{e}} \,. \tag{4.12}$$

The coefficient of numerical, artificial, or false diffusion (it goes by various uncomplimentary names!) is $\Gamma_{\mathrm{e}}^{\mathrm{num}} = (\rho u)_{\mathrm{e}} \Delta x/2$. This numerical diffusion is magnified in multidimensional problems if the flow is oblique to the grid; the truncation error then produces diffusion in the direction normal to the flow

as well as in the streamwise direction, a particularly serious type of error. Peaks or rapid variations in the variables will be smeared out and, since the rate of error reduction is only first order, very fine grids are required to obtain accurate solutions.

4.4.2 Linear Interpolation (CDS)

Another straightforward approximation for the value at CV-face center is linear interpolation between the two nearest nodes. At location 'e' on a Cartesian grid we have (see Figs. 4.2 and 4.3):

$$\phi_{\mathrm{e}} = \phi_{\mathrm{E}} \lambda_{\mathrm{e}} + \phi_{\mathrm{P}} (1 - \lambda_{\mathrm{e}}) \; , \tag{4.13}$$

where the linear interpolation factor λ_{e} is defined as:

$$\lambda_{\mathrm{e}} = \frac{x_{\mathrm{e}} - x_{\mathrm{P}}}{x_{\mathrm{E}} - x_{\mathrm{P}}} \; . \tag{4.14}$$

Equation (4.13) is second-order accurate as can be shown by using the Taylor series expansion of ϕ_E about the point x_P to eliminate the first derivative in Eq. (4.11). The result is:

$$\phi_{\mathrm{e}} = \phi_{\mathrm{E}} \lambda_{\mathrm{e}} + \phi_{\mathrm{P}} (1 - \lambda_{\mathrm{e}}) - \frac{(x_{\mathrm{e}} - x_{\mathrm{P}})(x_{\mathrm{E}} - x_{\mathrm{e}})}{2} \left(\frac{\partial^2 \phi}{\partial x^2} \right)_{\mathrm{P}} + H \; . \tag{4.15}$$

The leading truncation error term is proportional to the square of the grid spacing, on uniform or non-uniform grids.

As with all approximations of order higher than one, this scheme may produce oscillatory solutions. This is the simplest second-order scheme and is the one most widely used. It corresponds to the central-difference approximation of the first derivative in FD methods; hence the acronym CDS.

The assumption of a linear profile between the P and E nodes also offers the simplest approximation of the gradient, which is needed for the evaluation of diffusive fluxes:

$$\left(\frac{\partial \phi}{\partial x} \right)_{\mathrm{e}} \approx \frac{\phi_{\mathrm{E}} - \phi_{\mathrm{P}}}{x_{\mathrm{E}} - x_{\mathrm{P}}} \; . \tag{4.16}$$

By using Taylor series expansion around ϕ_{e} one can show that truncation error of the above approximation is:

$$\begin{aligned} \epsilon_\tau = & \frac{(x_{\mathrm{e}} - x_{\mathrm{P}})^2 - (x_{\mathrm{E}} - x_{\mathrm{e}})^2}{2\,(x_{\mathrm{E}} - x_{\mathrm{P}})} \left(\frac{\partial^2 \phi}{\partial x^2} \right)_{\mathrm{e}} - \\ & \frac{(x_{\mathrm{e}} - x_{\mathrm{P}})^3 + (x_{\mathrm{E}} - x_{\mathrm{e}})^3}{6\,(x_{\mathrm{E}} - x_{\mathrm{P}})} \left(\frac{\partial^3 \phi}{\partial x^3} \right)_{\mathrm{e}} + H \; . \end{aligned} \tag{4.17}$$

When the location 'e' is midway between P and E (for example on a uniform grid), the approximation is of second-order accuracy, since the first term on

the right-hand side vanishes and the leading error term is then proportional to $(\Delta x)^2$. When the grid is non-uniform, the leading error term is proportional to the product of Δx and the grid expansion factor minus unity. In spite of the formal first-order accuracy, the error reduction when the grid is refined is similar to that of a second-order approximation even on non-uniform grids. See Sect. 3.3.4 for a detailed explanation of this behavior.

4.4.3 Quadratic Upwind Interpolation (QUICK)

The next logical improvement is to approximate the variable profile between P and E by a parabola rather than a straight line. To construct a parabola we need to use data at one more point; in accord with the nature of convection, the third point is taken on the upstream side, i.e. W if the flow is from P to E (i.e. $u_x > 0$) or EE if $u_x < 0$, see Fig. 4.2. We thus obtain:

$$\phi_{\mathrm{e}} = \phi_{\mathrm{U}} + g_1(\phi_{\mathrm{D}} - \phi_{\mathrm{U}}) + g_2(\phi_{\mathrm{U}} - \phi_{\mathrm{UU}}) \;, \tag{4.18}$$

where D, U, and UU denote the downstream, the first upstream, and the second upstream node, respectively (E, P, and W or P, E, and EE, depending on the flow direction). The coefficients g_1 and g_2 can be expressed in terms of the nodal coordinates by:

$$g_1 = \frac{(x_{\mathrm{e}} - x_{\mathrm{U}})(x_{\mathrm{e}} - x_{\mathrm{UU}})}{(x_{\mathrm{D}} - x_{\mathrm{U}})(x_{\mathrm{D}} - x_{\mathrm{UU}})} \;; \quad g_2 = \frac{(x_{\mathrm{e}} - x_{\mathrm{U}})(x_{\mathrm{D}} - x_{\mathrm{e}})}{(x_{\mathrm{U}} - x_{\mathrm{UU}})(x_{\mathrm{D}} - x_{\mathrm{UU}})} \;.$$

For uniform grids, the coefficients of the three nodal values involved in the interpolation turn out to be: $\frac{3}{8}$ for the downstream point, $\frac{6}{8}$ for the first upstream node and $-\frac{1}{8}$ for the second upstream node. This scheme is somewhat more complex than the CDS scheme: it extends the computational molecule one more node in each direction (in 2D, the nodes EE, WW, NN and SS are included), and, on non-orthogonal and/or non-uniform grids, the expressions for the coefficients g_i are not simple. Leonard (1979) made this scheme popular and gave it the name QUICK (Quadratic Upwind Interpolation for Convective Kinematics).

This quadratic interpolation scheme has a third-order truncation error on both uniform and non-uniform grids. This can be shown by eliminating the second derivative from Eq. (4.15) using ϕ_{W}, which, on a uniform Cartesian grid with $u_x > 0$, leads to:

$$\phi_{\mathrm{e}} = \frac{6}{8}\,\phi_{\mathrm{P}} + \frac{3}{8}\,\phi_{\mathrm{E}} - \frac{1}{8}\,\phi_{\mathrm{W}} - \frac{3(\Delta x)^3}{48}\left(\frac{\partial^3 \phi}{\partial x^3}\right)_{\mathrm{P}} + H \;. \tag{4.19}$$

The first three terms on the right-hand side represent the QUICK approximation, while the last term is the principal truncation error. When this interpolation scheme is used in conjunction with the midpoint-rule approximation of the surface integral, the overall approximation is, however, still of

second-order accuracy (the accuracy of the quadrature approximation). Although the QUICK approximation is slightly more accurate than CDS, both schemes converge asymptotically in a second-order manner and the differences are rarely large.

4.4.4 Higher-Order Schemes

Interpolation of order higher than third makes sense only if the integrals are approximated using higher-order formulae. If one uses Simpson's rule in 2D for surface integrals, one has to interpolate with polynomials of at least degree three, which leads to interpolation errors of fourth order. For example, by fitting a polynomial

$$\phi(x) = a_0 + a_1\, x + a_2\, x^2 + a_3\, x^3 \tag{4.20}$$

through the values of ϕ at four nodes (two on either side of 'e': W, P, E and EE), one can determine the four coefficients a_i and find ϕ_e as a function of the nodal values. For a uniform Cartesian grid, the following expression is obtained:

$$\phi_e = \frac{27\,\phi_P + 27\,\phi_E - 3\,\phi_W - 3\,\phi_{EE}}{48} \; . \tag{4.21}$$

The same polynomial can be used to determine the derivative; we need only to differentiate it once to obtain:

$$\left(\frac{\partial \phi}{\partial x}\right)_e = a_1 + 2\,a_2\,x + 3\,a_3\,x^2 \; , \tag{4.22}$$

which, on a uniform Cartesian grid, produces:

$$\left(\frac{\partial \phi}{\partial x}\right)_e = \frac{27\,\phi_E - 27\,\phi_P + \phi_W - \phi_{EE}}{24\,\Delta x} \; . \tag{4.23}$$

The above approximation is sometimes called *fourth-order CDS*. Of course, both polynomials of higher degree and/or multi-dimensional polynomials can be used. Cubic splines, which ensure continuity of the interpolation function and its first two derivatives across the solution domain, can also be used (at some increase in cost).

Once the values of the variable and its derivative are obtained at the cell-face centers, one can interpolate on the cell faces to obtain values at the CV corners. This is not difficult to use with explicit methods but the fourth-order scheme based on Simpson's rule and polynomial interpolation produces too large a computational molecule for implicit treatment. One can avoid this complexity by using the *deferred-correction approach* described in Sect. 5.6.

Another approach is to use the techniques employed to derive the compact (Padé) schemes in FD methods. For example, one can obtain the coefficients

of the polynomial (4.20) by fitting it to the variable values and first derivatives at the two nodes on either side of the cell face. For a uniform Cartesian grid, the following expression for ϕ_e results:

$$\phi_e = \frac{\phi_P + \phi_E}{2} + \frac{\Delta x}{8}\left[\left(\frac{\partial \phi}{\partial x}\right)_P - \left(\frac{\partial \phi}{\partial x}\right)_E\right] . \tag{4.24}$$

The first term on the right-hand side of the above equation represents second-order approximation by linear interpolation; the second term represents an approximation of the second derivative which occurs in the leading truncation error term for linear interpolation, see Eq. (4.15).

The problem is that the derivatives at nodes P and E are not known and must themselves be approximated. However, even if we approximate the first derivatives by a second-order CDS, i.e.:

$$\left(\frac{\partial \phi}{\partial x}\right)_P = \frac{\phi_E - \phi_W}{2\,\Delta x} ; \quad \left(\frac{\partial \phi}{\partial x}\right)_E = \frac{\phi_{EE} - \phi_P}{2\,\Delta x} ,$$

the resulting approximation of the cell-face value retains the fourth-order accuracy of the polynomial:

$$\phi_e = \frac{\phi_P + \phi_E}{2} + \frac{\phi_P + \phi_E - \phi_W - \phi_{EE}}{16} + \mathcal{O}(\Delta x)^4 . \tag{4.25}$$

If we use as data the variable values on either side of the cell face and the derivative on the upstream side, we can fit a parabola. This leads to an approximation equivalent to the QUICK scheme described above:

$$\phi_e = \frac{3}{4}\phi_U + \frac{1}{4}\phi_D + \frac{\Delta x}{4}\left(\frac{\partial \phi}{\partial x}\right)_U . \tag{4.26}$$

The same approach can be used to obtain an approximation of the derivative at the cell-face center; from the derivative of the polynomial (4.20) we obtain:

$$\left(\frac{\partial \phi}{\partial x}\right)_e = \frac{\phi_E - \phi_P}{\Delta x} + \frac{\phi_E - \phi_P}{2\,\Delta x} - \frac{1}{4}\left[\left(\frac{\partial \phi}{\partial x}\right)_P + \left(\frac{\partial \phi}{\partial x}\right)_E\right] . \tag{4.27}$$

Obviously, the first term on the right-hand side is the second-order CDS approximation. The remaining terms represent a correction which increases the accuracy.

The problem with approximations (4.24), (4.26) and (4.27) is that they contain first derivatives at CV centers, which are not known. Although we can replace these by second-order approximations expressed in terms of the nodal variable values without destroying their order of accuracy, the resulting computational molecules will be much larger than we would like them to be. For example, in 2D, using Simpson's rule and fourth-order polynomial

interpolation, we find that each flux depends on 15 nodal values and the algebraic equation for one CV involves 25 values. The solution of the resulting equation system would be very expensive (see Chap. 5).

A way around this problem lies in the deferred-correction approach, that will be described in Sect. 5.6.

One should bear in mind that a higher-order approximation does not necessarily guarantee a more accurate solution on *any* single grid; high accuracy is achieved only when the grid is fine *enough* to capture all of the essential details of the solution; at what grid size this happens can be determined only by systematic grid refinement.

4.4.5 Other Schemes

A large number of approximations to the convective fluxes have been proposed; it is beyond the scope of this book to discuss all of them. The approach used above can be used to derive nearly all of them. We shall describe a few of them briefly.

One can approximate ϕ_e by linear extrapolation from two upstream nodes, leading to the so called *linear upwind scheme* (LUDS). This scheme is of second order accuracy but, as it is more complex than CDS and can produce unbounded solutions, the latter is a better choice.

Another approach, proposed by Raithby (1976), is to extrapolate from the upwind side, but along a streamline rather than a grid line (*skew upwind schemes*). First- and second-order schemes corresponding to the upwind and linear upwind schemes have been proposed. They have better accuracy than schemes based on extrapolation along grid lines. However, these schemes are very complex (there are many possible directions of flow) and a lot of interpolation is required. Since these schemes may produce oscillatory solutions when the grid is not sufficiently fine and are difficult to program, they have not found widespread use.

It is also possible to blend two or more different approximations. One example that saw a great deal of use in the 1970s and early 1980s is the hybrid scheme of Spalding (1972), which switches between UDS and CDS, depending on the local value of the Peclet number. Other researchers have proposed blending of lower and higher-order schemes to avoid unphysical oscillations, especially for compressible flows with shocks. Some of these ideas will be mentioned in Chap. 10. Blending may be used to improve the rate of convergence of some iterative solvers, as we shall show below.

4.5 Implementation of Boundary Conditions

Each CV provides one algebraic equation. Volume integrals are calculated in the same way for every CV, but fluxes through CV faces coinciding with

the domain boundary require special treatment. These boundary fluxes must either be known, or be expressed as a combination of interior values and boundary data. Since they do not give additional equations, they should not introduce additional unknowns. Since there are no nodes outside the boundary, these approximations must be based on one-sided differences or extrapolations.

Usually, convective fluxes are prescribed at the inflow boundary. Convective fluxes are zero at impermeable walls and symmetry planes, and are usually assumed to be independent of the coordinate normal to an outflow boundary; in this case, upwind approximations can be used. Diffusive fluxes are sometimes specified at a wall e.g. specified heat flux (including the special case of an adiabatic surface with zero heat flux) or boundary values of variables are prescribed. In such a case the diffusive fluxes are evaluated using one-sided approximations for normal gradients as outlined in Sect. 3.7. If the gradient itself is specified, it is used to calculate the flux, and an approximation for the flux in terms of nodal values can be used to calculate the boundary value of the variable. This will be demonstrated in an example below.

4.6 The Algebraic Equation System

By summing all the flux approximations and source terms, we produce an algebraic equation which relates the variable value at the center of the CV to the values at several neighbor CVs. The numbers of equations and unknowns are both equal to the number of CVs so the system is well-posed. The algebraic equation for a particular CV has the form (3.42), and the system of equations for the whole solution domain has the matrix form given by Eq. (3.43). When the ordering scheme of Sect. 3.8 is used, the matrix A has the form shown in Fig. 3.5. This is true only for structured grids with quadrilateral or hexahedral CVs; for other geometries, the matrix structure will be more complex (see Chap. 8 for details) but it will always be sparse. The maximum number of elements in any row is equal to the number of near neighbors for second order approximations. For higher-order approximations, it depends on the number of neighbors used in the scheme.

4.7 Examples

In order to demonstrate the FV method and to display some of the properties of the discretization methods presented above, we shall present two examples.

First consider the problem, illustrated in Fig. 4.4, of transport of a scalar quantity in a known velocity field. The latter is given by $u_x = x$ and $u_y = -y$, which represents the flow near a stagnation point. The streamlines are the

lines xy =const. and change direction with respect to the Cartesian grid. On the other hand, on any cell face the normal velocity component is constant so the error in the approximation of the convective flux depends only on the approximation used for ϕ_e. This aids in the analysis of the accuracy.

The scalar transport equation to be solved reads:

$$\int_S \rho\phi\boldsymbol{v}\cdot\boldsymbol{n}\,\mathrm{d}S = \int_S \Gamma\,\mathrm{grad}\,\phi\cdot\boldsymbol{n}\,\mathrm{d}S\;, \tag{4.28}$$

and the following boundary conditions are to be applied:

- $\phi = 0$ along the north (inlet) boundary;
- Linear variation of ϕ from $\phi = 0$ at $y = 1$ to $\phi = 1$ at $y = 0$ along the west boundary;
- Symmetry condition (zero gradient normal to boundary) on the south boundary;
- Zero gradient in the flow direction at outlet (east) boundary.

The geometry and the flow field are sketched in Fig. 4.4. We shall give the details of discretization for the 'e' face.
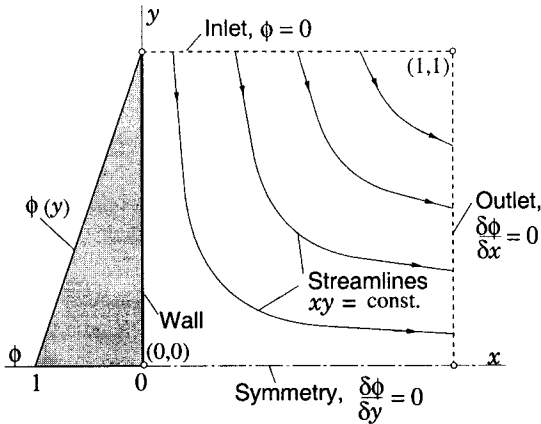
Fig. 4.4. Geometry and boundary conditions for the scalar transport in a stagnation point flow

The convective flux will be evaluated using the midpoint rule and either UDS or CDS interpolation. Since the normal velocity is constant along cell faces, we express the convective flux as a product of the mass flux and the mean value of ϕ:

$$F_\mathrm{e}^\mathrm{c} = \int_{S_\mathrm{e}} \rho\phi\boldsymbol{v}\cdot\boldsymbol{n}\,\mathrm{d}S \approx \dot{m}_\mathrm{e}\phi_\mathrm{e}\;, \tag{4.29}$$

where $\dot{m}_\mathrm{e}$ is the mass flux through the 'e' face:

$$\dot{m}_\mathrm{e} = \int_{S_\mathrm{e}} \rho \boldsymbol{v} \cdot \boldsymbol{n} \, \mathrm{d}S = (\rho u_x)_\mathrm{e} \Delta y \; . \tag{4.30}$$

Expression (4.30) is *exact* on any grid, since the velocity $u_{x,\mathrm{e}}$ is constant along the face. The flux approximation is then:

$$F_\mathrm{e}^\mathrm{c} = \begin{cases} \max(\dot{m}_\mathrm{e}, 0.)\, \phi_\mathrm{P} + \min(\dot{m}_\mathrm{e}, 0.)\, \phi_\mathrm{E} & \text{for UDS} \; ; \\ \dot{m}_\mathrm{e}(1 - \lambda_\mathrm{e})\, \phi_\mathrm{P} + \dot{m}_\mathrm{e}\lambda_\mathrm{e}\, \phi_\mathrm{E} & \text{for CDS} \; . \end{cases} \tag{4.31}$$

The linear interpolation coefficient λ_e is defined by Eq. (4.14). Analogous expressions for the fluxes through the other CV faces produce the following coefficients in the algebraic equation for the case of UDS:

$$\begin{aligned} &A_\mathrm{E}^\mathrm{c} = \min(\dot{m}_\mathrm{e}, 0.) \; ; \qquad A_\mathrm{W}^\mathrm{c} = \min(\dot{m}_\mathrm{w}, 0.) \; , \\ &A_\mathrm{N}^\mathrm{c} = \min(\dot{m}_\mathrm{n}, 0.) \; ; \qquad A_\mathrm{S}^\mathrm{c} = \min(\dot{m}_\mathrm{s}, 0.) \; , \\ &A_\mathrm{P}^\mathrm{c} = -(A_\mathrm{E}^\mathrm{c} + A_\mathrm{W}^\mathrm{c} + A_\mathrm{N}^\mathrm{c} + A_\mathrm{S}^\mathrm{c}) \; . \end{aligned} \tag{4.32}$$

For the CDS case, the coefficients are:

$$\begin{aligned} &A_\mathrm{E}^\mathrm{c} = \dot{m}_\mathrm{e}\lambda_\mathrm{e} \; ; \qquad A_\mathrm{W}^\mathrm{c} = \dot{m}_\mathrm{w}\lambda_\mathrm{w} \; , \\ &A_\mathrm{N}^\mathrm{c} = \dot{m}_\mathrm{n}\lambda_\mathrm{n} \; ; \qquad A_\mathrm{S}^\mathrm{c} = \dot{m}_\mathrm{s}\lambda_\mathrm{s} \; , \\ &A_\mathrm{P}^\mathrm{c} = -(A_\mathrm{E}^\mathrm{c} + A_\mathrm{W}^\mathrm{c} + A_\mathrm{N}^\mathrm{c} + A_\mathrm{S}^\mathrm{c}) \; . \end{aligned} \tag{4.33}$$

The expression for A_P^c follows from the continuity condition:

$$\dot{m}_\mathrm{e} + \dot{m}_\mathrm{w} + \dot{m}_\mathrm{n} + \dot{m}_\mathrm{s} = 0$$

which is satisfied by the velocity field. Note that $\dot{m}_\mathrm{w}$ and λ_w for the CV centered around node P are equal to $-\dot{m}_\mathrm{e}$ and $1 - \lambda_\mathrm{e}$ for the CV centered around node W, respectively. In a computer code the mass fluxes and interpolation factors are therefore calculated once and stored as $\dot{m}_\mathrm{e}$, $\dot{m}_\mathrm{n}$ and λ_e, λ_n for each CV.

The diffusive flux integral is evaluated using the midpoint rule and CDS approximation of the normal derivative; this is the simplest and most widely used approximation:

$$F_\mathrm{e}^\mathrm{d} = \int_{S_\mathrm{e}} \Gamma \operatorname{grad} \phi \cdot \boldsymbol{n} \, \mathrm{d}S \approx \left(\Gamma \frac{\partial \phi}{\partial x} \right)_\mathrm{e} \Delta y = \frac{\Gamma \Delta y}{x_\mathrm{E} - x_\mathrm{P}} (\phi_\mathrm{E} - \phi_\mathrm{P}) \; . \tag{4.34}$$

Note that $x_\mathrm{E} = \frac{1}{2}(x_{i+1} + x_i)$ and $x_\mathrm{P} = \frac{1}{2}(x_i + x_{i-1})$, see Fig. 4.2. The diffusion coefficient Γ is assumed constant; if not, it could be interpolated linearly between the nodal values at P and E. The contribution of the diffusion term to the coefficients of the algebraic equation are:

$$\begin{aligned} &A_\mathrm{E}^\mathrm{d} = -\frac{\Gamma \Delta y}{x_\mathrm{E} - x_\mathrm{P}} \; ; \qquad A_\mathrm{W}^\mathrm{d} = -\frac{\Gamma \Delta y}{x_\mathrm{P} - x_\mathrm{W}} \; , \\ &A_\mathrm{N}^\mathrm{d} = -\frac{\Gamma \Delta x}{y_\mathrm{N} - y_\mathrm{P}} \; ; \qquad A_\mathrm{S}^\mathrm{d} = -\frac{\Gamma \Delta x}{y_\mathrm{P} - y_\mathrm{S}} \; , \\ &A_\mathrm{P}^\mathrm{d} = -(A_\mathrm{E}^\mathrm{d} + A_\mathrm{W}^\mathrm{d} + A_\mathrm{N}^\mathrm{d} + A_\mathrm{S}^\mathrm{d}) \; . \end{aligned} \tag{4.35}$$

With same approximations applied to other CV faces, the integral equation becomes:

$$A_{\mathrm{W}}\phi_{\mathrm{W}} + A_{\mathrm{S}}\phi_{\mathrm{S}} + A_{\mathrm{P}}\phi_{\mathrm{P}} + A_{\mathrm{N}}\phi_{\mathrm{N}} + A_{\mathrm{E}}\phi_{\mathrm{E}} = Q_{\mathrm{P}} \ , \tag{4.36}$$

which represents the equation for a generic node P. The coefficients A_l are obtained by summing the convective and diffusive contributions, see Eqs. (4.32), (4.33) and (4.35):

$$A_l = A_l^{\mathrm{c}} + A_l^{\mathrm{d}} \tag{4.37}$$

where l represents any of the indices P, E, W, N, S. That A_{P} is equal to the negative sum of all neighbor coefficients is a feature of all conservative schemes and ensures that a uniform field is a solution of the discretized equations.

The above expressions are valid at all internal CVs. For CVs next to boundary, the boundary conditions require that the equations be modified somewhat. At the north and west boundaries, where ϕ is prescribed, the gradient in the normal direction is approximated using one-sided differences, e.g. at the west boundary:

$$\left(\frac{\partial \phi}{\partial x}\right)_{\mathrm{w}} \approx \frac{\phi_{\mathrm{P}} - \phi_{\mathrm{W}}}{x_{\mathrm{P}} - x_{\mathrm{W}}} \ , \tag{4.38}$$

where W denotes the boundary node whose location coincides with the cell-face center 'w'. This approximation is of first-order accuracy, but it is applied on a half-width CV. The product of the coefficient and the boundary value is added to the source term. For example, along the west boundary (CVs with index $i = 2$), $A_{\mathrm{W}}\phi_{\mathrm{W}}$ is added to Q_{P} and the coefficient A_{W} is set to zero. The same applies to the coefficient A_{N} at the north boundary.

At the south boundary, the normal gradient of ϕ is zero which, when the above approximation is applied, means that the boundary values are equal to the values at CV centers. Thus, for cells with index $j = 2$, $\phi_{\mathrm{S}} = \phi_{\mathrm{P}}$ and the algebraic equation for those CVs is modified to:

$$(A_{\mathrm{P}} + A_{\mathrm{S}})\,\phi_{\mathrm{P}} + A_{\mathrm{N}}\phi_{\mathrm{N}} + A_{\mathrm{W}}\phi_{\mathrm{W}} + A_{\mathrm{E}}\phi_{\mathrm{E}} = Q_{\mathrm{P}} \ , \tag{4.39}$$

which requires adding A_{S} to A_{P} and then setting $A_{\mathrm{S}} = 0$. The zero-gradient condition at the outlet (east) boundary is implemented in a similar way.

We now turn to the results. The isolines of ϕ calculated on a 40×40 CV uniform grid using CDS for the convective fluxes with two values of Γ: 0.001 and 0.01 ($\rho = 1.0$) are presented in Fig. 4.5. We see that transport by diffusion across the flow is much stronger for higher Γ, as expected.

In order to assess the accuracy of the prediction, we monitor the total flux of ϕ through the west boundary, at which ϕ is prescribed. This quantity is obtained by summing diffusive fluxes over all CV faces along this boundary, which are approximated by Eqs. (4.34) and (4.38). Figure 4.6 shows the variation of the flux as the grid is refined for the UDS and CDS discretizations of
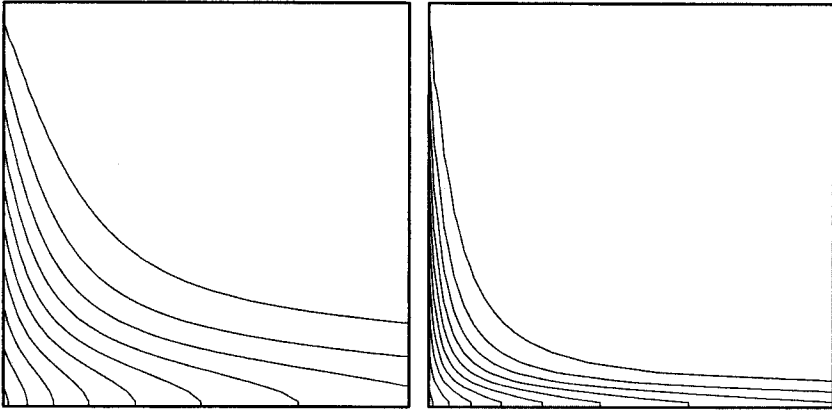
Fig. 4.5. Isolines of ϕ, from 0.05 to 0.95 with step 0.1 (top to bottom), for $\Gamma = 0.01$ (left) and $\Gamma = 0.001$ (right)

the convective fluxes; the diffusive fluxes are always discretized using CDS. The grid was refined from 10 × 10 CV to 320 × 320 CV. On the coarsest grid, the CDS does not produce a meaningful solution for $\Gamma = 0.001$; convection dominates and, on such a coarse grid, the rapid change in ϕ over short distance near the west boundary (see Fig. 4.5) results in oscillations so strong that most iterative solvers fail to converge (the local cell Peclet numbers, $Pe = \rho u_x \Delta x / \Gamma$, range between 10 and 100 on this grid). (A converged solution could probably be obtained with the aid of deferred correction but it would be very inaccurate.) As the grid is refined, the CDS result converges monotonically towards a grid-independent solution. On the 40 × 40 CV grid the local Peclet numbers range from 2.5 to 25, but there are no oscillations in the solution, as can be seen in Fig. 4.5.

The UDS solution does not oscillate on any grid, as expected. The convergence is, however, not monotonic: the flux on the two coarsest grids lies below the converged value; it is too high on the next grid and then approaches the correct result monotonically. By assuming second-order convergence of the CDS scheme, we estimated the grid-independent solution via Richardson extrapolation (see Sect. 3.9 for details) and were able to determine the error in each solution. The errors are plotted vs. normalized grid size ($\Delta x = 1$ for the coarsest grid) in Fig. 4.6 for both UDS and CDS. The expected slopes for first- and second-order schemes are also shown. The CDS error curve has the slope expected of a second-order scheme. The UDS error shows irregular behavior on the first three grids. From the fourth grid onwards the error curve approaches the expected slope. The solution on the grid with 320 × 320 CV is still in error by over 1%; CDS produces a more accurate result on the 80 × 80 grid!

Another popular test case is the convection of a step profile in a uniform flow oblique to grid lines, see Fig. 4.7. It can be solved using the method de-
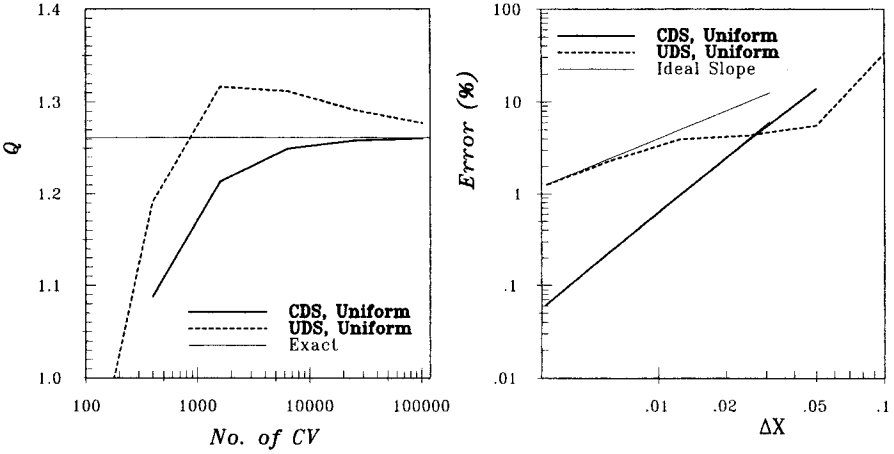
Fig. 4.6. Convergence of total flux of ϕ through the west wall (left) and the error in computed flux as a function of grid spacing, for $\Gamma = 0.001$

scribed above by adjusting the boundary conditions (prescribed values of ϕ at west and south boundaries, outflow conditions at north and east boundaries). We show below the results obtained using the UDS and CDS discretizations.

Since diffusion is not present in this case, the equation to be solved is (in differential form):

$$u_x \frac{\partial \phi}{\partial x} + u_y \frac{\partial \phi}{\partial y} = 0 \; . \tag{4.40}$$

For this case, the UDS on a uniform grid in both directions gives the very simple equation:

$$u_x \frac{\phi_{\mathrm{P}} - \phi_{\mathrm{W}}}{\Delta x} + u_y \frac{\phi_{\mathrm{P}} - \phi_{\mathrm{S}}}{\Delta y} = 0 \; , \tag{4.41}$$

which is readily solved in a sequential manner without iteration. On the other hand, CDS gives a zero value for the coefficient on the main diagonal, A_{P}, making solution difficult. Most iterative solvers would fail to converge for this problem; however, by using the deferred correction approach described above, it is possible to obtain the solution.

If the flow is parallel to x-coordinate, both schemes give the correct result: the profile is simply convected downstream. When the flow is oblique to grid lines, UDS produces a smeared step profile at any downstream cross-section, while CDS produces oscillations. In Fig. 4.8 we show the profile of ϕ at $x = 0.45$ for the the case when the flow is at 45° to the grid ($u_x = u_y$), obtained on a uniform 10 × 10 CV grid. The same figure shows the profile at $x = 0.475$ for the same case obtained on a uniform 20 × 20 CV grid. The effect of numerical diffusion is clearly seen in the UDS solution; little
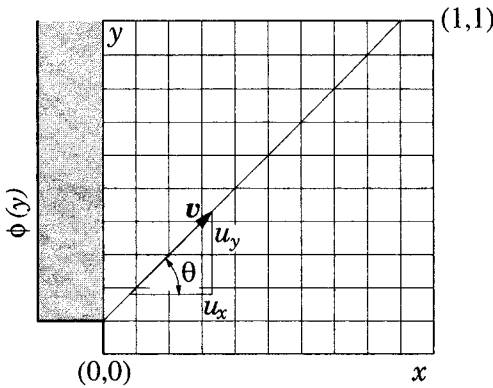
Fig. 4.7. Geometry and boundary conditions for convection of a step profile in a uniform flow oblique to grid lines

improvement is found in the solution on the refined grid. On the other hand, CDS produces a profile with the proper steepness, but it oscillates. Local grid refinement would help localize and, perhaps even remove, the oscillations as will be discussed in Chap. 11. The oscillations could also be removed by locally introducing numerical diffusion (e.g. by blending CDS with UDS). This is sometimes done in compressible flows near shocks.
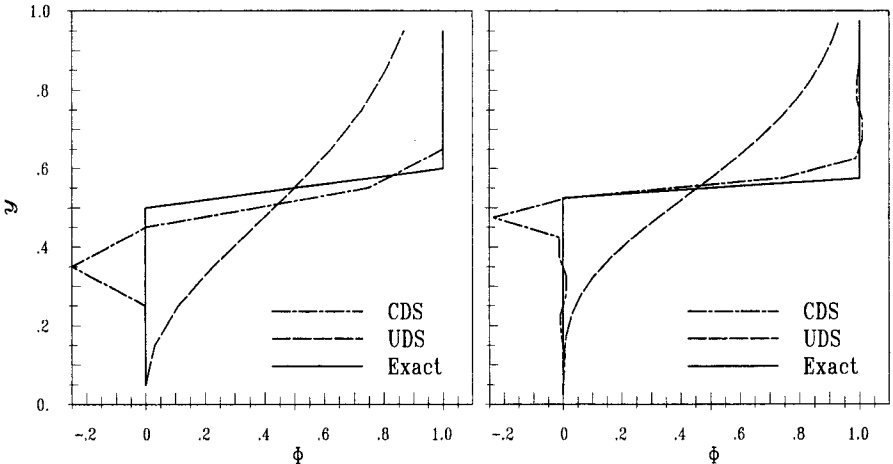
Fig. 4.8. Profile of ϕ at $x = 0.45$, calculated on a 10×10 CV grid (left), and at $x = 0.475$, calculated on a 20×20 CV grid (right)

One can show that the UDS method embodied in Eq. (4.41) is more nearly solving the convection-diffusion problem:

$$u_x \frac{\partial \phi}{\partial x} + u_y \frac{\partial \phi}{\partial y} = u_x \, \Delta x \, \frac{\partial^2 \phi}{\partial x^2} + u_y \, \Delta y \, \frac{\partial^2 \phi}{\partial y^2} \tag{4.42}$$

than the original Eq. (4.40). Equation (4.42) is called the *modified equation* for this problem. By transforming this equation into coordinates parallel and perpendicular to the flow, one can show that the effective diffusivity in the normal direction is:

$$\Gamma_{\text{eff}} = U \sin\theta \cos\theta(\Delta x \cos\theta + \Delta y \sin\theta) , \tag{4.43}$$

where U is the magnitude of the velocity and θ is the angle of the flow with respect to the x-direction. A similar and widely quoted result was derived by de Vahl Davis and Mallinson (1972).

To sum up our findings, we have shown:

- High-order schemes oscillate on coarse grids but converge to an accurate solution more rapidly than low order schemes as the grid is refined.
- First-order UDS is inaccurate and should not be used. This scheme is mentioned because it is still used in some commercial codes. Users should be aware that high accuracy cannot be obtained on affordable grids with this method, especially in 3D. It introduces a large diffusive error in both the streamwise and normal directions.
- CDS is the simplest scheme of second-order accuracy and offers a good compromise among accuracy, simplicity and efficiency.

5. Solution of Linear Equation Systems

5.1 Introduction

In the previous two chapters we showed how the convection-diffusion equation may be discretized using FD and FV methods. In either case, the result of the discretization process is a system of algebraic equations, which are linear or non-linear according to the nature of the partial differential equation(s) from which they are derived. In the non-linear case, the discretized equations must be solved by an iterative technique that involves guessing a solution, linearizing the equations about that solution, and improving the solution; the process is repeated until a converged result is obtained. So, whether the equations are linear or not, efficient methods for solving linear systems of algebraic equations are needed.

The matrices derived from partial differential equations are always sparse, i.e. most of their elements are zero. Some methods for solving the equations that arise when structured grids are used will be described below; all of the non-zero elements of the matrices then lie on a small number of well-defined diagonals; we may take advantage of this structure. Some of the methods are applicable to matrices arising from unstructured grids as well.

The structure of the coefficient matrix for a 2D problem discretized with a five point approximation (upwind or central difference) is shown in Fig. 3.5. The algebraic equation for one CV or grid node is given by Eq. (3.42), and the matrix version of the complete problem is given by Eq. (3.43), see Sect. 3.8, which is repeated here:

$$A\phi = \boldsymbol{Q} \; . \tag{5.1}$$

In addition to describing some of the better solution methods for linear algebraic systems representing discretized partial differential equations, we shall discuss the solution of non-linear systems of equations in this chapter. However, we begin with linear equations. It is assumed that the reader has had some contact with methods for solving linear systems so the descriptions are brief.

5.2 Direct Methods

The matrix A is assumed to be very sparse. In fact, the most complicated matrix we shall encounter is a banded matrix of block type; this greatly simplifies the task of solution but we shall briefly review methods for general matrices as methods for sparse matrices are closely related to them. For the description of methods designed to deal with full matrices, use of full-matrix notation (as opposed to the diagonal notation introduced earlier) is more sensible and will be adopted.

5.2.1 Gauss Elimination

The basic method for solving linear systems of algebraic equations is Gauss elimination. Its basis is the systematic reduction of large systems of equations to smaller ones. In this procedure, the elements of the matrix are modified but, as the dependent variable names do not change, it is convenient to describe the method in terms of the matrix alone:

$$A = \begin{pmatrix} A_{11} & A_{12} & A_{13} & \dots & A_{1n} \\ A_{21} & A_{22} & A_{23} & \dots & A_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & A_{n3} & \dots & A_{nn} \end{pmatrix} . \quad (5.2)$$

The heart of the algorithm is the technique for eliminating A_{21} i.e., replacing it with a zero. This is accomplished by multiplying the first equation (first row of the matrix) by A_{21}/A_{11} and subtracting it from the second row or equation; in the process, all of the elements in the second row of the matrix are modified as is the second element of the forcing vector on the right hand side of the equation. The other elements of the first column of the matrix, $A_{31}, A_{41}, \dots, A_{n1}$ are treated similarly; for example, to eliminate A_{i1}, the first row of the matrix is multiplied by A_{i1}/A_{11} and subtracted from the ith row. By systematically proceeding down the first column of the matrix, all of the elements below A_{11} are eliminated. When this process is complete, none of the equations $2, 3, \dots, n$ contain the variable ϕ_1; they are a set of $n - 1$ equations for the variables $\phi_2, \phi_3, \dots, \phi_n$. The same procedure is then applied to this smaller set of equations – all of the elements below A_{22} in the second column are eliminated.

This procedure is carried out for columns $1, 2, 3, \dots, n-1$. After this process is complete, the original matrix has been replaced by an upper triangular one:

$$U = \begin{pmatrix} A_{11} & A_{12} & A_{13} & \dots & A_{1n} \\ 0 & A_{22} & A_{23} & \dots & A_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & A_{nn} \end{pmatrix} . \quad (5.3)$$

All of the elements except those in the first row differ from those in the original matrix A. As the elements of the original matrix will never be needed again, it is efficient to store the modified elements in place of the original ones. (In the rare case requiring that the original matrix be saved, a copy can be created prior to starting the elimination procedure.)

This portion of the algorithm just described is called *forward elimination.* The elements on the right hand side of the equation, Q_i, are also modified in this procedure.

The upper triangular system of equations resulting from forward elimination is easily solved. The last equation contains only one variable, ϕ_n and is readily solved:

$$\phi_n = \frac{Q_n}{A_{nn}} . \tag{5.4}$$

The next to last equation contains only ϕ_{n-1} and ϕ_n and, once ϕ_n is known, it can be solved for ϕ_{n-1}. Proceeding upward in this manner, each equation is solved in turn; the ith equation yields ϕ_i:

$$\phi_i = \frac{Q_i - \sum_{k=i+1}^{n} A_{ik}\phi_k}{A_{ii}} . \tag{5.5}$$

The right hand side is calculable because all of the ϕ_k appearing in the sum have already been evaluated. In this way, all of the variables may be computed. The part of the Gauss elimination algorithm which starts with the triangular matrix and computes the unknowns is called *back substitution.*

It is not difficult to show that, for large n, the number of operations required to solve a linear system of n equations by Gauss elimination is proportional to $n^3/3$. The bulk of this effort is in the forward elimination phase; the back substitution requires only $n^2/2$ arithmetic operations and is much less costly than the forward elimination. Gauss elimination is thus expensive but, for full matrices, it is as good as any method available. The high cost of Gauss elimination provides incentive to search for more efficient special solvers for matrices such as the sparse ones arising from the discretization of differential equations.

For large systems that are not sparse, Gauss elimination is susceptible to accumulation of errors (see Golub and van Loan, 1990) which makes it unreliable if not modified. The addition of *pivoting* or interchange of rows in order to make the pivot elements (the diagonal elements that appear in the denominators) as large as possible, keeps the error growth in check. Fortunately, for sparse matrices, error accumulation is rarely a problem so this issue is not important here.

Gauss elimination does not vectorize or parallelize well and is rarely used without modification in CFD problems.

5.2.2 LU Decomposition

A number of variations on Gauss elimination have been proposed. Most are of little interest here. One variant of value to CFD is LU decomposition. It is presented without derivation.

We have seen that, in Gauss elimination, forward elimination reduces a full matrix to an upper triangular one. This process can be carried out in a more formal manner by multiplying the original matrix A by a lower triangular matrix. By itself, this is of little interest but, as the inverse of a lower triangular matrix is also lower triangular, this result shows that any matrix A, subject to some limitations that can be ignored here, can be factored into the product of lower (L) and upper (U) triangular matrices:

$$A = LU \; . \tag{5.6}$$

To make the factorization unique, we require that the diagonal elements of L, L_{ii}, all be unity; alternatively, one could require the diagonal elements of U to be unity.

What makes this factorization useful is that it is easily constructed. The upper triangular matrix U is precisely the one produced by the forward phase of Gauss elimination. Furthermore, the elements of L are the multiplicative factors (e.g. A_{ji}/A_{ii}) used in the elimination process. This allows the factorization to be constructed by a minor modification of Gauss elimination. Furthermore, the elements of L and U can be stored where the elements of A were.

The existence of this factorization allows the solution of the system of equations (5.1) in two stages. With the definition:

$$U\boldsymbol{\phi} = \boldsymbol{Y} \; , \tag{5.7}$$

the system of equations (5.1) becomes:

$$L\boldsymbol{Y} = \boldsymbol{Q} \; . \tag{5.8}$$

The latter set of equations can be solved by a variation of the method used in the backward substitution phase of Gauss elimination in which one starts from the top rather than the bottom of the system. Once Eq. (5.8) has been solved for $\boldsymbol{Y}$, Eq. (5.7), which is identical to the triangular system solved in the back substitution phase of Gauss elimination, can be solved for $\boldsymbol{\phi}$.

The advantage of LU factorization over Gauss elimination is that the factorization can be performed without knowing the vector $\boldsymbol{Q}$. As a result, if many systems involving the same matrix are to be solved, considerable savings can be obtained by performing the factorization first; the systems can then be solved as required. As we shall see below, variations on LU factorization are the basis of some of the better iterative methods of solving systems of linear equations; this is the principal reason for introducing it here.

5.2.3 Tridiagonal Systems

When ordinary differential equations (1D problems) are finite differenced, for example, with the CDS approximation, the resulting algebraic equations have an especially simple structure. Each equation contains only the variables at its own node and its immediate left and right neighbors:

$$A_{\mathrm{W}}^{i}\phi_{i-1} + A_{\mathrm{P}}^{i}\phi_{i} + A_{\mathrm{E}}^{i}\phi_{i+1} = Q_i \ . \tag{5.9}$$

The corresponding matrix A has non-zero terms only on its main diagonal (represented by A_{P}) and the diagonals immediately above and below it (represented by A_{E} and A_{W}, respectively). Such a matrix is called *tridiagonal*; systems containing tridiagonal matrices are especially easy to solve. The matrix elements are best stored as three $n \times 1$ arrays.

Gauss elimination is especially easy for tridiagonal systems: only one element needs to be eliminated from each row during the forward elimination process. When the algorithm has reached the ith row, only A_{P}^{i} needs to be modified; the new value is:

$$A_{\mathrm{P}}^{i} = A_{\mathrm{P}}^{i} - \frac{A_{\mathrm{W}}^{i} A_{\mathrm{E}}^{i-1}}{A_{\mathrm{P}}^{i-1}} \ , \tag{5.10}$$

where this equation is to be understood in the programmer's sense that the result is stored in place of the original A_{P}^{i}. The forcing term is also modified:

$$Q_i^* = Q_i - \frac{A_{\mathrm{W}}^{i} Q_{i-1}^*}{A_{\mathrm{P}}^{i-1}} \ . \tag{5.11}$$

The back substitution part of the method is also simple. The ith variable is computed from:

$$\phi_i = \frac{Q_i^* - A_{\mathrm{E}}^{i}\phi_{i+1}}{A_{\mathrm{P}}^{i}} \ . \tag{5.12}$$

This tridiagonal solution method is sometimes called the *Thomas Algorithm* or the Tridiagonal Matrix Algorithm (TDMA). It is easily programmed (a FORTRAN code requires only eight executable lines) and, more importantly, the number of operations is proportional to n, the number of unknowns, rather than the n^3 of full matrix Gauss elimination. In other words, the cost per unknown is independent of the number of unknowns, which is almost as good a scaling as one could desire. The low cost suggests that this algorithm be employed whenever possible. Many solution methods take advantage of the low cost of this method by reducing the problem to one involving tridiagonal matrices.

5.2.4 Cyclic Reduction

There are cases even more special that allow still greater cost reduction than that offered by TDMA. An interesting example is provided by systems in which the matrix is not only tridiagonal but all of the elements on each diagonal are identical. The *cyclic reduction* method can be used to solve such systems with a cost per variable that actually decreases as the system becomes larger. Let us see how that is possible.

Suppose that, in the system (5.9), the coefficients $A^i_{\mathrm{W}}, A^i_{\mathrm{P}}$ and A^i_{E} are independent of the index i; we may then drop the index. Then, for even values of i, we multiply row $i-1$ by $A_{\mathrm{W}}/A_{\mathrm{P}}$ and subtract it from row i. Then we multiply row $i+1$ by $A_{\mathrm{E}}/A_{\mathrm{P}}$ and subtract it from row i. This eliminates the elements to the immediate left and right of the main diagonal in the even numbered rows but replaces the zero element two columns to the left of the main diagonal by $-A^2_{\mathrm{W}}/A_{\mathrm{P}}$ and the zero element two columns to the right of the main diagonal by $-A^2_{\mathrm{E}}/A_{\mathrm{P}}$; the diagonal element becomes $A_{\mathrm{P}} - 2A_{\mathrm{W}}A_{\mathrm{E}}/A_{\mathrm{P}}$. Because the elements in every even row are the same, the calculation of the new elements needs to be done only once; this is where the savings come from.

At the completion of these operations the even numbered equations contain only even indexed variables and constitute a set of $n/2$ equations for these variables; considered as a separate system, these equations are tridiagonal and the elements on each diagonal of the reduced matrix are again equal. In other words, the reduced set of equations has the same form as the original one but is half the size. It can be further reduced in the same way. If the number of equations in the original set is a power of two (or certain other convenient numbers), the method can be continued until only one equation remains; the latter is solved directly. The remaining variables can then be found by a variant of back substitution.

One can show that the cost of this method is proportional to $\log_2 n$, so that the cost per variable decreases with the number of variables. Although the method may seem rather specialized, there are CFD applications in which it plays a role. These are flows in very regular geometries such as the rectangular boxes which are used, for example, in direct or large eddy simulations of turbulence and in some meteorological applications.

In these applications, cyclic reduction and related methods provide the basis for methods of solving elliptic equations such as Laplace and Poisson equations directly, that is, non-iteratively. Since the solutions are also exact in the sense that they contain no iteration error, this method is invaluable whenever it can be used.

Cyclic reduction is closely related to the fast Fourier transform which is also used to solve elliptic equations in simple geometries. Fourier methods may also be used for evaluating derivatives, as was shown in Sect. 3.10.

5.3 Iterative Methods

5.3.1 Basic Concept

Any system of equations can be solved by Gauss elimination or LU decomposition. Unfortunately, the triangular factors of sparse matrices are not sparse, so the cost of these methods is quite high. Furthermore, the discretization error is usually much larger than the accuracy of the computer arithmetic so there is no reason to solve the system that accurately. Solution to somewhat more accuracy than that of the discretization scheme suffices.

This leaves an opening for iterative methods. They are used out of necessity for non-linear problems, but they are just as valuable for sparse linear systems. In an iterative method, one guesses a solution, and uses the equation to systematically improve it. If each iteration is cheap and the number of iterations is small, an iterative solver may cost less than a direct method. In CFD problems this is usually the case.

Consider the matrix problem represented by Eq. (5.1) which might result from FD or FV approximation of a flow problem. After n iterations we have an approximate solution ϕ^n which does not satisfy these equations exactly. Instead, there is a non-zero residual ρ^n:

$$A\boldsymbol{\phi}^n = \boldsymbol{Q} - \boldsymbol{\rho}^n \; . \tag{5.13}$$

By subtracting this equation from Eq. (5.1), we obtain a relation between the iteration error defined by:

$$\boldsymbol{\epsilon}^n = \boldsymbol{\phi} - \boldsymbol{\phi}^n \; , \tag{5.14}$$

where $\boldsymbol{\phi}$ is the converged solution, and the residual:

$$A\boldsymbol{\epsilon}^n = \boldsymbol{\rho}^n \; . \tag{5.15}$$

The purpose of the iteration procedure is to drive the residual to zero; in the process, ϵ also becomes zero. To see how this can be done, consider an iterative scheme for a linear system; such a scheme can be written:

$$M\boldsymbol{\phi}^{n+1} = N\boldsymbol{\phi}^n + \boldsymbol{B} \; . \tag{5.16}$$

An obvious property that must be demanded of an iterative method is that the converged result satisfies Eq. (5.1). Since, by definition, at convergence, $\boldsymbol{\phi}^{n+1} = \boldsymbol{\phi}^n = \boldsymbol{\phi}$, we must have:

$$A = M - N \quad \text{and} \quad \boldsymbol{B} = \boldsymbol{Q} \; , \tag{5.17}$$

or, more generally,

$$PA = M - N \quad \text{and} \quad \boldsymbol{B} = P\boldsymbol{Q} \; , \tag{5.18}$$

where P is a non-singular *pre-conditioning matrix.*

An alternative version of this iterative method may be obtained by subtracting $M\boldsymbol{\phi}^n$ from each side of Eq. (5.16). We obtain:

$$M(\phi^{n+1} - \phi^n) = \boldsymbol{B} - (M - N)\phi^n \quad \text{or} \quad M\delta^n = \rho^n , \tag{5.19}$$

where $\delta^n = \phi^{n+1} - \phi^n$ is called the *correction* or update and is an approximation to the iteration error.

For an iterative method to be effective, solving the system (5.16) must be cheap and the method must converge rapidly. Inexpensive iteration requires that computation of $N\phi^n$ and solution of the system must both be easy to perform. The first requirement is easily met; since A is sparse, N is also sparse, and computation of $N\phi^n$ is simple. The second requirement means that the iteration matrix M must be easily inverted; from a practical point of view, M should be diagonal, tridiagonal, triangular, or, perhaps, block tridiagonal or triangular; another possibility is described below. For rapid convergence, M should be a good approximation to A, making $N\phi$ small in some sense. This is discussed further below.

5.3.2 Convergence

As we have noted, rapid convergence of an iterative method is key to its effectiveness. Here we give a simple analysis that is useful in understanding what determines the convergence rate and provides insight into how to improve it.

To begin, we derive the equation that determines the behavior of the iteration error. To find it, we recall that, at convergence, $\phi^{n+1} = \phi^n = \phi$, so that the converged solution obeys the equation:

$$M\phi = N\phi + \boldsymbol{B} . \tag{5.20}$$

Subtracting this equation from Eq. (5.16) and using the definition (5.14) of the iteration error, we find:

$$M\epsilon^{n+1} = N\epsilon^n \tag{5.21}$$

or

$$\epsilon^{n+1} = M^{-1}N\epsilon^n . \tag{5.22}$$

The iterative method converges if $\lim_{n\to\infty} \epsilon^n = 0$. The critical role is played by the eigenvalues λ_k and eigenvectors ψ^k of the iteration matrix $M^{-1}N$ which are defined by:

$$M^{-1}N\psi^k = \lambda_k\psi^k , \quad k = 1, \ldots, K , \tag{5.23}$$

where K is the number of equations (grid points). We assume that the eigenvectors form a complete set i.e. a basis for $\boldsymbol{R}^n$, the vector space of all n-component vectors. If that is so, the initial error may be expressed in terms of them:

$$\epsilon^0 = \sum_{k=1}^{K} a_k\psi^k , \tag{5.24}$$

where a_k is a constant. Then the iterative procedure (5.22) yields:

$$\epsilon^1 = M^{-1}N\epsilon^0 = M^{-1}N\sum_{k=1}^{K} a_k\psi^k = \sum_{k=1}^{K} a_k\lambda_k\psi^k \tag{5.25}$$

and, by induction, it is not difficult to show that

$$\epsilon^n = \sum_{k=1}^{K} a_k(\lambda_k)^n\psi^k \ . \tag{5.26}$$

It is clear that, if ϵ^n is to become zero when n is large, the necessary and sufficient condition is that all of the eigenvalues must be less than unity in magnitude. In particular, this must be true of the largest eigenvalue, whose magnitude is called the *spectral radius* of the matrix $M^{-1}N$. In fact, after a number of iterations, the terms in Eq. (5.26) that contain eigenvalues of small magnitude become very small and only the term containing the largest eigenvalue (which we can take to be λ_1 and assume to be unique) remains:

$$\epsilon^n \sim a_1(\lambda_1)^n\psi^1 \ . \tag{5.27}$$

If convergence is defined as the reduction of the iteration error below some tolerance δ, we require:

$$a_1(\lambda_1)^n \approx \delta \ . \tag{5.28}$$

Taking the logarithm of both sides of this equation, we find an expression for the required number of iterations:

$$n \approx \frac{\ln\left(\dfrac{\delta}{a_1}\right)}{\ln\lambda_1} \ . \tag{5.29}$$

We see that, if the spectral radius is very close to unity, the iterative procedure will converge very slowly.

As a simple (trivial might be more descriptive) example consider the case of a single equation (for which one would never dream of using an iterative method). Suppose we want to solve:

$$ax = b \tag{5.30}$$

and we use the iterative method (note that $m = a + n$ and p is the iteration counter):

$$mx^{p+1} = nx^p + b \ . \tag{5.31}$$

Then the error obeys the scalar equivalent of Eq. (5.22):

$$\epsilon^{p+1} = \frac{n}{m}\epsilon^p \ . \tag{5.32}$$

We see that the error is reduced quickly if n/m is small i.e. if n is small, which means that $m \approx a$. In constructing iterative methods for systems, we shall find that an analogous result holds: the more closely M approximates A, the more rapid the convergence.

In an iterative method it is important to be able to estimate the iteration error in order to decide when to stop iterating. Calculation of the eigenvalues of the iteration matrix is difficult (it is often not explicitly known), so approximations have to be used. We shall describe some methods of estimating the iteration error and criteria for stopping iterations later in this chapter.

5.3.3 Some Basic Methods

In the simplest method, the Jacobi method, M is a diagonal matrix whose elements are the diagonal elements of A. For the five-point discretization of Laplace equation, if each iteration is begun at the lower left (southwest) corner of the domain and we use the geographic notation introduced above, the method is:

$$\phi_{\mathrm{P}}^{n+1} = \frac{Q_{\mathrm{P}} - A_{\mathrm{S}}\phi_{\mathrm{S}}^{n} - A_{\mathrm{W}}\phi_{\mathrm{W}}^{n} - A_{\mathrm{N}}\phi_{\mathrm{N}}^{n} - A_{\mathrm{E}}\phi_{\mathrm{E}}^{n}}{A_{\mathrm{P}}} \ . \tag{5.33}$$

It may be shown that, for convergence, this method requires a number of iterations proportional to the square of the number of grid points in one direction. This means that it is more expensive than a direct solver so there is little reason to use it.

In the Gauss-Seidel method, M is the lower triangular portion of A. As it is a special case of the SOR method given below, we shall not give the equations separately. It converges twice as fast as the Jacobi method but this is not enough of an improvement to be useful.

One of the better methods is an accelerated version of the Gauss-Seidel method called *successive over-relaxation* or SOR, which we shall describe below. For an introduction and analysis of the Jacobi and Gauss-Seidel methods, see an introductory text on numerical methods such as Ferziger (1998) or Press et al. (1987).

If each iteration is begun at the lower left (southwest) corner of the domain and we again use the geographic notation, the SOR method can be written:

$$\phi_{\mathrm{P}}^{n+1} = \omega\, \frac{Q_{\mathrm{P}} - A_{\mathrm{S}}\phi_{\mathrm{S}}^{n+1} - A_{\mathrm{W}}\phi_{\mathrm{W}}^{n+1} - A_{\mathrm{N}}\phi_{\mathrm{N}}^{n} - A_{\mathrm{E}}\phi_{\mathrm{E}}^{n}}{A_{\mathrm{P}}} + (1-\omega)\phi_{\mathrm{P}}^{n} \ , \tag{5.34}$$

where ω is the over-relaxation factor, which must be greater than 1 for acceleration, and n is the iteration counter. There is theory to guide the selection of the optimum over-relaxation factor for simple problems such as Laplace equation in a rectangular domain but it is hard to apply that theory to more complex problems; fortunately, the behavior of the method is usually similar to that found in the simple case. Generally, the larger the grid, the larger the

optimum over-relaxation factor. For values of ω less than the optimum, the convergence is monotonic and the rate of convergence increases as ω increases. When the optimum ω is exceeded, the convergence rate deteriorates and the convergence is oscillatory. This knowledge can be used to search for the optimum over-relaxation factor. When the optimum over-relaxation factor is used, the number of iterations is proportional to the number of grid points in one direction, a substantial improvement over the methods mentioned above. For $\omega = 1$, SOR reduces to the Gauss-Seidel method.

5.3.4 Incomplete LU Decomposition: Stone's Method

We make two observations. LU decomposition is an excellent general-purpose linear systems solver but it cannot take advantage of the sparseness of a matrix. In an iterative method, if M is a good approximation to A, rapid convergence results. These observations lead to the idea of using an approximate LU factorization of A as the iteration matrix M i.e.:

$$M = LU = A + N , \tag{5.35}$$

where L and U are both sparse and N is small.

A version of this method for symmetric matrices, known as *incomplete Cholesky* factorization is often used in conjunction with conjugate gradient methods. Since the matrices that arise from discretizing convection-diffusion problems or the Navier-Stokes equations are not symmetric, this method cannot be applied to them. An asymmetric version of this method, called *incomplete* LU factorization or ILU, is possible but has not found widespread use. In the ILU method one proceeds as in LU decomposition but, for every element of the original matrix A that is zero, the corresponding element of L or U is set to zero. This factorization is not exact, but the product of these factors can be used as the matrix M of the iterative method. This method converges rather slowly.
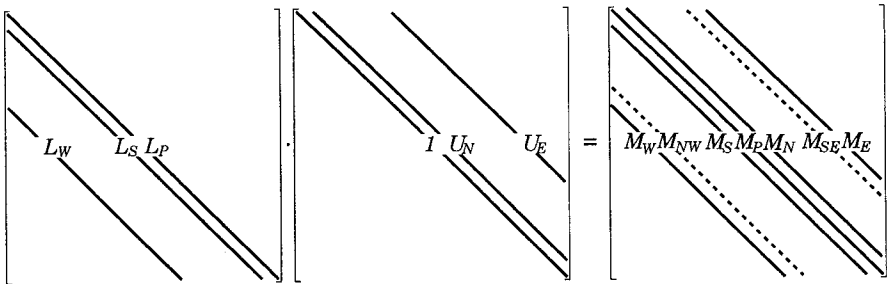
Fig. 5.1. Schematic presentation of the matrices L and U and the product matrix M; diagonals of M not found in A are shown by dashed lines

Another incomplete lower-upper decomposition method, which has found use in CFD, was proposed by Stone (1968). This method, also called the *strongly implicit procedure* (SIP), is specifically designed for algebraic equations that are discretizations of partial differential equations and does not apply to generic systems of equations.

We shall describe the SIP method for the five-point computational molecule, i.e. a matrix with the structure shown in Fig. 3.5. The same principles can be used to construct solvers for 7-point (in three-dimensions) and 9-point (for two-dimensional non-orthogonal grids) computational molecules.

As in ILU, the L and U matrices have non-zero elements only on diagonals on which A has non-zero elements. The product of lower and upper triangular matrices with these structures has more non-zero diagonals than A. For the standard five-point molecule there are two more diagonals (corresponding to nodes NW and SE or NE and SW, depending on the ordering of the nodes in the vector), and for seven-point molecules in 3D, there are six more diagonals. For the ordering of nodes used in this book for 2D problems, the extra two diagonals correspond to the nodes NW and SE (see Table 3.2 for the correspondence of the grid indices (i, j) and the one-dimensional storage location index l).

To make these matrices unique, every element on the main diagonal of U is set to unity. Thus five sets of elements (three in L, two in U) need to be determined. For matrices of the form shown in Fig. 5.1, the rules of matrix multiplication give the elements of the product of L and U, $M = LU$:

$$
\begin{aligned}
M^l_{\mathrm{W}} &= L^l_{\mathrm{W}} \\
M^l_{\mathrm{NW}} &= L^l_{\mathrm{W}} U^{l-N_j}_{\mathrm{N}} \\
M^l_{\mathrm{S}} &= L^l_{\mathrm{S}} \\
M^l_{\mathrm{P}} &= L^l_{\mathrm{W}} U^{l-N_j}_{\mathrm{E}} + L^l_{\mathrm{S}} U^{l-1}_{\mathrm{N}} + L^l_{\mathrm{P}} \\
M^l_{\mathrm{N}} &= U^l_{\mathrm{N}} L^l_{\mathrm{P}} \\
M^l_{\mathrm{SE}} &= L^l_{\mathrm{S}} U^{l-1}_{\mathrm{E}} \\
M^l_{\mathrm{E}} &= U^l_{\mathrm{E}} L^l_{\mathrm{P}}
\end{aligned}
\tag{5.36}
$$

We wish to select L and U, such that M is as good an approximation to A as possible. At minimum, N must contain the two diagonals of M that correspond to zero diagonals of A, see Eq. (5.36). An obvious choice is to let N have non-zero elements on only these two diagonals, and force the other diagonals of M to equal the corresponding diagonals of A. This is possible; in fact, this is the standard ILU method mentioned earlier. Unfortunately, this method converges slowly.

Stone (1968) recognized that convergence can be improved by allowing N to have non-zero elements on the diagonals corresponding to all seven non-zero diagonals of LU. The method is most easily derived by considering the

vector $M\phi$:

$$
\begin{aligned}
(M\phi)_{\mathrm{P}} = & M_{\mathrm{P}}\phi_{\mathrm{P}} + M_{\mathrm{S}}\phi_{\mathrm{S}} + M_{\mathrm{N}}\phi_{\mathrm{N}} + M_{\mathrm{E}}\phi_{\mathrm{E}} + M_{\mathrm{W}}\phi_{\mathrm{W}} + \\
& M_{\mathrm{NW}}\phi_{\mathrm{NW}} + M_{\mathrm{SE}}\phi_{\mathrm{SE}} \ .
\end{aligned} \tag{5.37}
$$

The last two terms are the 'extra' ones. Each term in this equation corresponds to a diagonal of $M = LU$.

The matrix N must contain the two 'extra' diagonals of M, and we want to choose the elements on the remaining diagonals so that $N\phi \approx \mathbf{0}$ or, in other words,

$$
\begin{aligned}
& N_{\mathrm{P}}\phi_{\mathrm{P}} + N_{\mathrm{N}}\phi_{\mathrm{N}} + N_{\mathrm{S}}\phi_{\mathrm{S}} + \\
& N_{\mathrm{E}}\phi_{\mathrm{E}} + N_{\mathrm{W}}\phi_{\mathrm{W}} + M_{\mathrm{NW}}\phi_{\mathrm{NW}} + M_{\mathrm{SE}}\phi_{\mathrm{SE}} \approx 0 \ .
\end{aligned} \tag{5.38}
$$

This requires that the contribution of the two 'extra' terms in the above equation be nearly canceled by the contribution of other diagonals. In other words, Eq. (5.38) should reduce to the following expression:

$$
M_{\mathrm{NW}}(\phi_{\mathrm{NW}} - \phi^*_{\mathrm{NW}}) + M_{\mathrm{SE}}(\phi_{\mathrm{SE}} - \phi^*_{\mathrm{SE}}) \approx 0 \ , \tag{5.39}
$$

where ϕ^*_{NW} and ϕ^*_{SE} are approximations to ϕ_{NW} and ϕ_{SE}.

Stone's key idea is that, since the equations approximate an elliptic partial differential equation, the solution can be expected to be smooth. This being so, ϕ^*_{NW} and ϕ^*_{SE} can be approximated in terms of the values of ϕ at nodes corresponding to the diagonals of A. Stone proposed the following approximation (other approximations are possible, see Schneider and Zedan, 1981, for an example):

$$
\begin{aligned}
\phi^*_{\mathrm{NW}} &\approx \alpha(\phi_{\mathrm{W}} + \phi_{\mathrm{N}} - \phi_{\mathrm{P}}) \\
\phi^*_{\mathrm{SE}} &\approx \alpha(\phi_{\mathrm{S}} + \phi_{\mathrm{E}} - \phi_{\mathrm{P}})
\end{aligned} \tag{5.40}
$$

If $\alpha = 1$, these are second order accurate interpolations but Stone found that stability requires $\alpha < 1$. These approximations are based on the connection to partial differential equations and make little sense for generic algebraic equations.

If these approximations are substituted into Eq. (5.39) and the result is equated to Eq. (5.38), we obtain all elements of N as linear combinations of M_{NW} and M_{SE}. The elements of M, Eq. (5.36), can now be set equal to the sum of elements of A and N. The resulting equations are not only sufficient to determine all of the elements of L and U, but they can be solved in sequential order beginning at the southwest corner of the grid:

$$
\begin{aligned}
L^l_{\mathrm{W}} &= A^l_{\mathrm{W}} / \left(1 + \alpha U^{l-N_j}_{\mathrm{N}}\right) \\
L^l_{\mathrm{S}} &= A^l_{\mathrm{S}} / \left(1 + \alpha U^{l-1}_{\mathrm{E}}\right) \\
L^l_{\mathrm{P}} &= A^l_{\mathrm{P}} + \alpha\left(L^l_{\mathrm{W}} U^{l-N_j}_{\mathrm{N}} + L^l_{\mathrm{S}} U^{l-1}_{\mathrm{E}}\right) - L^l_{\mathrm{W}} U^{l-N_j}_{\mathrm{E}} - L^l_{\mathrm{S}} U^{l-1}_{\mathrm{N}}
\end{aligned} \tag{5.41}
$$

$$U_{\mathrm{N}}^{l} = \left(A_{\mathrm{N}}^{l} - \alpha L_{\mathrm{W}}^{l} U_{\mathrm{N}}^{l-N_j}\right) / L_{\mathrm{P}}^{l}$$

$$U_{\mathrm{E}}^{l} = \left(A_{\mathrm{E}}^{l} - \alpha L_{\mathrm{S}}^{l} U_{\mathrm{E}}^{l-1}\right) / L_{\mathrm{P}}^{l}$$

The coefficients must be calculated in this order. For nodes next to boundaries, any matrix element that carries the index of a boundary node is understood to be zero. Thus, along the west boundary ($i = 2$), elements with index $l - N_j$ are zero; along the south boundary ($j = 2$), elements with index $l - 1$ are zero; along the north boundary ($j = N_j - 1$), elements with index $l + 1$ are zero; finally, along the east boundary ($i = N_i - 1$), elements with index $l + N_j$ are zero.

We now turn to solving the system of equations with the aid of this approximate factorization. The equation relating the update to the residual is (see Eq. (5.19)):

$$LU\boldsymbol{\delta}^{n+1} = \boldsymbol{\rho}^{n} \ . \tag{5.42}$$

The equations are solved as in in generic LU decomposition. Multiplication of the above equation by L^{-1} leads to:

$$U\boldsymbol{\delta}^{n+1} = L^{-1}\boldsymbol{\rho}^{n} = \boldsymbol{R}^{n} \ . \tag{5.43}$$

$\boldsymbol{R}^n$ is easily computed:

$$R^{l} = \left(\rho^{l} - L_{\mathrm{S}}^{l} R^{l-1} - L_{\mathrm{W}}^{l} R^{l-N_j}\right) / L_{\mathrm{P}}^{l} \ . \tag{5.44}$$

This equation is to be solved by marching in the order of increasing l. When the computation of $\boldsymbol{R}$ is complete, we need to solve Eq. (5.43):

$$\delta^{l} = R^{l} - U_{\mathrm{N}}^{l} \delta^{l+1} - U_{\mathrm{E}}^{l} \delta^{l+N_j} \tag{5.45}$$

in order of decreasing index l.

In the SIP method, the elements of the matrices L and U need be calculated only once, prior to the first iteration. On subsequent iterations, we need calculate only the residual, then $\boldsymbol{R}$ and finally $\boldsymbol{\delta}$, by solving the two triangular systems.

Stone's method usually converges in a small number of iterations. The rate of convergence can be improved by varying α from iteration to iteration (and point to point). These methods converge in fewer iterations but they require the factorization to be redone each time α is changed. Since computing L and U is as expensive as an iteration with a given decomposition, it is usually more efficient overall to keep α fixed.

Stone's method can be generalized to yield an efficient solver for the nine-diagonal matrices that arise when compact difference approximations are applied in two dimensions and for the seven-diagonal matrices that arise when central differences are used in three dimensions. A 3D (7-point) vectorized version is given by Leister and Perić (1994); two 9-point versions for 2D problems are described by Schneider and Zedan (1981) and Perić (1987). Computer codes for five-diagonal (2D) and seven-diagonal (3D) matrices are

available via Internet; see appendix for details. The performance of SIP for a model problem will be presented in Sect. 5.8.

Unlike other methods, Stone's method is both a good iterative technique in its own right and a good basis for conjugate gradient methods (where it is called a preconditioner) and multigrid methods (where it is used as a smoother). These methods will be described below.

5.3.5 ADI and Other Splitting Methods

A common method of solving elliptic problems is to add a term containing the first time derivative to the equation and solve the resulting parabolic problem until a steady state is reached. At that point, the time derivative is zero and the solution satisfies the original elliptic equation. Many iterative methods of solving elliptic equations, including most of those already described, can be interpreted in this way. In this section we present a method whose connection to parabolic equations is so close that it might not have been discovered if one were thinking only of elliptic equations.

Considerations of stability require methods for parabolic equations to be implicit in time. In two or three dimensions, this requires solution of a two or three dimensional elliptic problem at each time step; the cost can be enormous but it can be reduced considerably by using the *alternating direction implicit* or ADI method. We give only the simplest such method in two dimensions and a variant. ADI is the basis for many other methods; for more details of some of these methods see Hageman and Young (1981).

Suppose we want to solve Laplace equation in two dimensions. Adding a time derivative to it converts it to the heat equation in two dimensions:

$$\frac{\partial \phi}{\partial t} = \Gamma \left(\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} \right) . \tag{5.46}$$

If this equation is discretized using the trapezoid rule in time (called Crank-Nicolson when applied to partial differential equations; see next chapter), and central differences are used to approximate the spatial derivatives on a uniform grid, we obtain:

$$\frac{\phi^{n+1} - \phi^n}{\Delta t} = \frac{\Gamma}{2} \left[\left(\frac{\delta^2 \phi^n}{\delta x^2} + \frac{\delta^2 \phi^n}{\delta y^2} \right) + \left(\frac{\delta^2 \phi^{n+1}}{\delta x^2} + \frac{\delta^2 \phi^{n+1}}{\delta y^2} \right) \right] , \tag{5.47}$$

where we have used the shorthand notation:

$$\left(\frac{\delta^2 \phi}{\delta x^2} \right)_{i,j} = \frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{(\Delta x)^2} ,$$

$$\left(\frac{\delta^2 \phi}{\delta y^2} \right)_{i,j} = \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{(\Delta y)^2}$$

for the spatial finite differences. Rearranging Eq. (5.47), we find that, at time step $n+1$, we have to solve the system of equations:

$$\left(1 - \frac{\Gamma \Delta t}{2} \frac{\delta^2}{\delta x^2}\right) \left(1 - \frac{\Gamma \Delta t}{2} \frac{\delta^2}{\delta y^2}\right) \phi^{n+1} = \left(1 + \frac{\Gamma \Delta t}{2} \frac{\delta^2}{\delta x^2}\right) \left(1 + \frac{\Gamma \Delta t}{2} \frac{\delta^2}{\delta y^2}\right) \phi^{n} - \frac{(\Gamma \Delta t)^2}{4} \frac{\delta^2}{\delta x^2} \left[\frac{\delta^2 (\phi^{n+1} - \phi^n)}{\delta y^2}\right] . \tag{5.48}$$

As $\phi^{n+1} - \phi^n \approx \Delta t \, \partial\phi/\partial t$, the last term is proportional to $(\Delta t)^3$ for small Δt. Since the FD approximation is of second order, for small Δt, the last term is small compared to the discretization error and may be neglected. The remaining equation can be factored into two simpler equations:

$$\left(1 - \frac{\Gamma \Delta t}{2} \frac{\delta^2}{\delta x^2}\right) \phi^* = \left(1 + \frac{\Gamma \Delta t}{2} \frac{\delta^2}{\delta y^2}\right) \phi^n , \tag{5.49}$$

$$\left(1 - \frac{\Gamma \Delta t}{2} \frac{\delta^2}{\delta y^2}\right) \phi^{n+1} = \left(1 + \frac{\Gamma \Delta t}{2} \frac{\delta^2}{\delta x^2}\right) \phi^* . \tag{5.50}$$

Each of these systems of equations is a set of tridiagonal equations that can be solved with the efficient TDMA method; this requires no iteration and is much cheaper than solving Eq. (5.47). Either Eq. (5.49) or (5.50), as a method in its own right, is only first-order accurate in time and conditionally stable but the combined method is second-order accurate and unconditionally stable! The family of methods based on these ideas are known as *splitting* or *approximate factorization* methods; a wide variety of them has been developed.

Neglect of the third-order term, which is essential to the factorization, is justified only when the time step is small. So, although the method is unconditionally stable, it may not be accurate in time if the time step is large. For elliptic equations, the objective is to obtain the steady state solution as quickly as possible; this is best accomplished with the largest possible time step. However, the factorization error becomes large when the time step is large so the method loses some of its effectiveness. In fact, there is an optimum time step which gives the most rapid convergence. When this time step is used, the ADI method is very efficient – it converges in a number of iterations proportional to the number of points in one direction.

A better strategy uses different time steps for several iterations in a cyclic fashion. This approach can make the number of iterations for convergence proportional to the square root of the number of grid points in one direction, making ADI an excellent method.

Equations which involve the convection and source terms require some generalization of this method. In CFD, the pressure or pressure-correction

equation is of the above type and the variants of the method just described are often used to solve it. ADI methods are very commonly used when solving compressible flow problems. They are also well adapted to parallel computation.

The method described in this section takes advantage of the structure of the matrix which is, in turn, due to the use of a structured grid. However, closer inspection of the development shows that the basis of the method is an *additive decomposition* of the matrix:

$$A = H + V \; , \tag{5.51}$$

where H is the matrix representing the terms contributed by the second derivative with respect to x and V, the terms coming from the second derivative in the y-direction.

There is no reason why other additive decompositions cannot be used. One useful suggestion is to consider the additive LU decomposition:

$$A = L + U \; . \tag{5.52}$$

This is different from the multiplicative LU decomposition of Sect. 5.2.2. With this decomposition, Eqs. (5.49) and (5.50) are replaced by:

$$\begin{aligned} (I - L\,\Delta t)\phi^* &= (I + U\,\Delta t)\phi^n \; , \\ (I - U\,\Delta t)\phi^{n+1} &= (I + L\,\Delta t)\phi^* \; . \end{aligned} \tag{5.53}$$

Each of these steps is essentially a Gauss-Seidel iteration. The rate of convergence of this method is similar that of the ADI method given above. It also has the very important advantage that it does not rely on the structure of the grid or of the matrix and may therefore be applied to problems on unstructured grids as well as structured ones. However, it does not parallelize as well as the HV version of ADI.

5.3.6 Conjugate Gradient Methods

In this section, we present a class of methods based on techniques for solving non-linear equations. Non-linear solvers can be grouped into two broad categories: Newton-like methods and global methods. The former converge very quickly if an accurate estimate of the solution is available but may fail catastrophically if the initial guess is far from the exact solution. 'Far' is a relative term; it is different for each equation. One cannot determine whether an estimate is 'close enough' except by trial and error. Global methods are guaranteed to find the solution (if one exists) but are not very fast. Combinations of the two types of methods are often used; global methods are used initially and followed by Newton-like methods as convergence is approached.

Many global methods are descent methods. These methods begin by converting the original system of equations into a minimization problem. Suppose that the set of equations to be solved is given by Eq. (5.1) and that the matrix

A is symmetric and its eigenvalues are positive; such a matrix is called *positive definite.* (Most matrices associated with problems in fluid mechanics are not symmetric or positive definite so we will need to generalize this method later.) For positive definite matrices, solving the system of equations (5.1) is equivalent to the problem of finding the minimum of

$$F = \frac{1}{2}\boldsymbol{\phi}^T A \boldsymbol{\phi} - \boldsymbol{\phi}^T \boldsymbol{Q} = \frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n} A_{ij}\phi_i\phi_j - \sum_{i=1}^{n}\phi_i Q_i \tag{5.54}$$

with respect to all the ϕ_i; this may be verified by taking the derivative of F with respect to each variable and setting it equal to zero. A way to convert the original system into a minimization problem that does not require positive definiteness is to take the sum of the squares of all of the equations but this introduces additional difficulties.

The oldest and best known method for seeking the minimum of a function is *steepest descents.* The function F may be thought of as a surface in (hyper)-space. Suppose we have some starting guess which may be represented as a point in that (hyper-)space. At that point, we find the steepest downward path on the surface; it lies in the direction opposite to the gradient of the function. We then search for the lowest point on that line. By construction, it has a lower value of F than the starting point; in this sense, the new estimate is closer to the solution. The new value is then used as the starting point for the next iteration and the process is continued until it converges. Unfortunately, while it is guaranteed to converge, steepest descents often converges very slowly. If the contour plot of the magnitude of the function F has a narrow valley, the method tends to oscillate back and forth across that valley and many steps may be required to find the solution. In other words, the method tends to use the same search directions over and over again.

Many improvements have been suggested. The easiest ones require the new search directions to be as different from the old ones as possible. Among these is the *conjugate gradient* method. We shall give only the general idea and a description of the algorithm here; a more complete presentation can be found in the book by Golub and van Loan (1990).

The conjugate gradient method is based on a remarkable discovery: it is possible to minimize a function with respect to several directions simultaneously while searching in one direction at a time. This is made possible by a clever choice of directions. We shall describe this for the case of two directions; suppose we wish to find values of α_1 and α_2 in

$$\boldsymbol{\phi} = \boldsymbol{\phi}^0 + \alpha_1 \boldsymbol{p}^1 + \alpha_2 \boldsymbol{p}^2 \tag{5.55}$$

which minimize F; that is, we try to minimize F in the $\boldsymbol{p}^1 - \boldsymbol{p}^2$ plane. This problem can be reduced to the problem of minimizing with respect to $\boldsymbol{p}^1$ and $\boldsymbol{p}^2$ individually provided that the two directions are conjugate in the following sense:

$$\boldsymbol{p}^1 \cdot A \boldsymbol{p}^2 = 0 \,. \tag{5.56}$$

This property is akin to orthogonality; the vectors $\boldsymbol{p}^1$ and $\boldsymbol{p}^2$ are said to be conjugate with respect to the matrix A, which gives the method its name. A detailed proof of this statement and others cited below can be found in the book of Golub and van Loan (1990).

This property can be extended to any number of directions. In the conjugate gradient method, each new search direction is required to be conjugate to all the preceding ones. If the matrix is non-singular, as is the case in nearly all engineering problems, the directions are guaranteed to be linearly independent. Consequently, if exact (no round-off error) arithmetic were employed, the conjugate gradient method would converge exactly when the number of iterations is equal to the size of the matrix. This number can be quite large and, in practice, exact convergence is not achieved due to arithmetic errors. It is therefore wiser to regard the conjugate gradient method as an iterative method.

While the conjugate gradient method guarantees that the error is reduced on each iteration, the size of the reduction depends on the search direction. It is not unusual for this method to reduce the error only slightly for a number of iterations and then find a direction that reduces the error by an order of magnitude or more in one iteration.

It can be shown that the rate of convergence of this method depends on the *condition number* κ of the matrix where

$$\kappa = \frac{\lambda_{\text{max}}}{\lambda_{\text{min}}} \tag{5.57}$$

and λ_{max} and λ_{min} are the largest and smallest eigenvalues of the matrix. The condition numbers of matrices that arise in CFD problems are usually approximately the square of the maximum number of grid points in any direction. With 100 grid points in each direction, the condition number should be about 10^4 and the standard conjugate gradient method would converge slowly. Although the conjugate gradient method is significantly faster than steepest descents for a given condition number, this basic method is not very useful.

This method can be improved by replacing the problem whose solution we seek by another one with the same solution but a smaller condition number. For obvious reasons, this is called *preconditioning.* One way to precondition the problem is to pre-multiply the equation by another (carefully chosen) matrix. As this would destroy the symmetry of the matrix, the preconditioning must take the following form:

$$C^{-1}AC^{-1}C\phi = C^{-1}\boldsymbol{Q} \,. \tag{5.58}$$

The conjugate gradient method is applied to the matrix $C^{-1}AC^{-1}$ i.e. to the modified problem (5.58). If this is done and the residual form of the iterative method is used, the following algorithm results (for a detailed derivation, see

Golub and van Loan, 1990). In this description, $\boldsymbol{\rho}^k$ is the residual at the kth iteration, $\boldsymbol{p}^k$ is the kth search direction, $\boldsymbol{z}^k$ is an auxiliary vector and α_k and β_k are parameters used in constructing the new solution, residual, and search direction. The algorithm can be summarized as follows:

- Initialize by setting: $k = 0$, $\boldsymbol{\phi}^0 = \boldsymbol{\phi}_{\text{in}}$, $\boldsymbol{\rho}^0 = \boldsymbol{Q} - A\boldsymbol{\phi}_{\text{in}}$, $\boldsymbol{p}^0 = \boldsymbol{0}$, $s_0 = 10^{30}$
- Advance the counter: $k = k + 1$
- Solve the system: $M\boldsymbol{z}^k = \boldsymbol{\rho}^{k-1}$
- Calculate: $s^k = \boldsymbol{\rho}^{k-1} \cdot \boldsymbol{z}^k$

  $\beta^k = s^k / s^{k-1}$

  $\boldsymbol{p}^k = \boldsymbol{z}^k + \beta^k \boldsymbol{p}^{k-1}$

  $\alpha^k = s_k / (\boldsymbol{p}^k \cdot A\boldsymbol{p}^k)$

  $\boldsymbol{\phi}^k = \boldsymbol{\phi}^{k-1} + \alpha^k \boldsymbol{p}^k$

  $\boldsymbol{\rho}^k = \boldsymbol{\rho}^{k-1} - \alpha^k A\boldsymbol{p}^k$
- Repeat until convergence.

This algorithm involves solving a system of linear equations at the first step. The matrix involved is $M = C^{-1}$ where C is the pre-conditioning matrix, which is in fact never actually constructed. For the method to be efficient, M must be easy to invert. The choice of M used most often is incomplete Cholesky factorization of A but in tests it was found that if $M = LU$ where L and U are the factors used in Stone's SIP method, faster convergence is obtained. Examples will be presented below.

5.3.7 Biconjugate Gradients and CGSTAB

The conjugate gradient method given above is applicable only to symmetric systems; the matrices obtained by discretizing the Poisson equation are often symmetric (examples are the heat conduction equation and pressure or pressure-correction equations to be introduced in Chap. 7). To apply the method to systems of equations that are not necessary symmetric (for example, any convection/diffusion equation), we need to convert an asymmetric problem to a symmetric one. There are a couple of ways of doing this of which the following is perhaps the simplest. Consider the system:

$$\begin{pmatrix} 0 & A \\ A^T & 0 \end{pmatrix} \cdot \begin{pmatrix} \boldsymbol{\psi} \\ \boldsymbol{\phi} \end{pmatrix} = \begin{pmatrix} \boldsymbol{Q} \\ \boldsymbol{0} \end{pmatrix} . \tag{5.59}$$

This system can be decomposed into two subsystems. The first is the original system; the second involves the transpose matrix and is irrelevant. (If there were a need to do so, one could solve a system of equations involving the transpose matrix at little extra cost.) When the pre-conditioned conjugate gradient method is applied to this system, the following method, called *biconjugate gradients*, results:

- Initialize by setting: $k = 0$, $\boldsymbol{\phi}^0 = \boldsymbol{\phi}_{\text{in}}$, $\boldsymbol{\rho}^0 = \boldsymbol{Q} - A\boldsymbol{\phi}_{\text{in}}$, $\overline{\boldsymbol{\rho}}^0 = \boldsymbol{Q} - A^T\boldsymbol{\phi}_{\text{in}}$, $\boldsymbol{p}^0 = \overline{\boldsymbol{p}}^0 = \boldsymbol{0}$, $s_0 = 10^{30}$

- Advance the counter: $k = k + 1$
- Solve the systems: $M\boldsymbol{z}^k = \boldsymbol{\rho}^{k-1}$, $M^T\overline{\boldsymbol{z}}^k = \overline{\boldsymbol{\rho}}^{k-1}$
- Calculate: $s^k = \boldsymbol{z}^k \cdot \overline{\boldsymbol{\rho}}^{k-1}$

$$\beta^k = s^k / s^{k-1}$$
$$\boldsymbol{p}^k = \boldsymbol{z}^k + \beta^k \boldsymbol{p}^{k-1}$$
$$\overline{\boldsymbol{p}}^k = \overline{\boldsymbol{z}}^k + \beta^k \overline{\boldsymbol{p}}^{k-1}$$
$$\alpha^k = s^k / (\overline{\boldsymbol{p}}^k A \boldsymbol{p}^k)$$
$$\boldsymbol{\phi}^k = \boldsymbol{\phi}^{k-1} + \alpha^k \boldsymbol{p}^k$$
$$\boldsymbol{\rho}^k = \boldsymbol{\rho}^{k-1} - \alpha^k A \boldsymbol{p}^k$$
$$\overline{\boldsymbol{\rho}}^k = \overline{\boldsymbol{\rho}}^{k-1} - \alpha^k A^T \overline{\boldsymbol{p}}^k$$

- Repeat until convergence.

The above algorithm was published by Fletcher (1976). It requires almost exactly twice as much effort per iteration as the standard conjugate gradient method but converges in about the same number of iterations. It has not been widely used in CFD applications but it appears to be very robust (meaning that it handles a wide range of problems without difficulty).

Other variants of the biconjugate gradient method type which are more stable and robust have been developed. We mention here the CGS (conjugate gradient squared) algorithm, proposed by Sonneveld (1989); CGSTAB (CGS stabilized), proposed by Van den Vorst and Sonneveld (1990) and another version by Van den Vorst (1992); and GMRES, proposed by Saad and Schultz (1986). All of these can be applied to non-symmetric matrices and to both structured and unstructured grids. We give below the CGSTAB algorithm without formal derivation:

- Initialize by setting: $k = 0$, $\boldsymbol{\phi}^0 = \boldsymbol{\phi}_{\text{in}}$, $\boldsymbol{\rho}^0 = \boldsymbol{Q} - A\boldsymbol{\phi}_{\text{in}}$, $\boldsymbol{u}^0 = \boldsymbol{p}^0 = \boldsymbol{0}$
- Advance the counter $k = k + 1$ and calculate:

$$\beta^k = \boldsymbol{\rho}^0 \cdot \boldsymbol{\rho}^{k-1}$$
$$\omega^k = (\beta^k \gamma^{k-1}) / (\alpha^{k-1} \beta^{k-1})$$
$$\boldsymbol{p}^k = \boldsymbol{\rho}^{k-1} + \omega^k (\boldsymbol{p}^{k-1} - \alpha^{k-1} \boldsymbol{u}^{k-1})$$

- Solve the system: $M\boldsymbol{z} = \boldsymbol{p}^k$
- Calculate: $\boldsymbol{u}^k = A\boldsymbol{z}$

$$\gamma^k = \beta^k / (\boldsymbol{u}^k \cdot \boldsymbol{\rho}^0)$$
$$\boldsymbol{w} = \boldsymbol{\rho}^{k-1} - \gamma^k \boldsymbol{u}^k$$

- Solve the system: $M\boldsymbol{y} = \boldsymbol{w}$
- Calculate: $\boldsymbol{v} = A\boldsymbol{y}$

$$\alpha^k = (\boldsymbol{v} \cdot \boldsymbol{\rho}^k) / (\boldsymbol{v} \cdot \boldsymbol{v})$$
$$\boldsymbol{\phi}^k = \boldsymbol{\phi}^{k-1} + \gamma^k \boldsymbol{z} + \alpha^k \boldsymbol{y}$$
$$\boldsymbol{\rho}^k = \boldsymbol{w} - \alpha^k \boldsymbol{v}$$

- Repeat until convergence.

Note that $\boldsymbol{u}$, $\boldsymbol{v}$, $\boldsymbol{w}$, $\boldsymbol{y}$ and $\boldsymbol{z}$ are auxiliary vectors and have nothing to do with the velocity vector or the coordinates y and z. The algorithm can be programmed as given above; computer codes for the conjugate gradient method with incomplete Cholesky preconditioning (ICCG, for symmetric matrices,

both 2D and 3D versions) and the 3D CGSTAB solver are available via Internet; see appendix for details. A biconjugate gradient solver with incomplete Cholesky preconditioning for nine-point schemes in 2D is also provided.

5.3.8 Multigrid Methods

The final method for solving linear systems to be discussed here is the multigrid method. The basis for the multigrid concept is an observation about iterative methods. Their rate of convergence depends on the eigenvalues of the iteration matrix associated with the method. In particular, the eigenvalue(s) with largest magnitude (the *spectral radius* of the matrix) determines how rapidly the solution is reached; see Sect. 5.3.2. The eigenvector(s) associated with this eigenvalue(s) determines the spatial distribution of the iteration error and varies considerably from method to method. Let us briefly review the behavior of these entities for some of the methods presented above. The properties are given for Laplace equation; most of them generalize to other elliptic partial differential equations.

For Laplace equation, the two largest eigenvalues of the Jacobi method are real and of opposite sign. One eigenvector represents a smooth function of the spatial coordinates, the other, a rapidly oscillating function. The iteration error for the Jacobi method is thus a mixture of very smooth and very rough components; this makes acceleration difficult. On the other hand, the Gauss-Seidel method has a single real positive largest eigenvalue with an eigenvector that makes the iteration error a smooth function of the spatial coordinates.

The largest eigenvalues of the SOR method with optimum over-relaxation factor lie on a circle in the complex plane and there are a number of them; consequently, the error behaves in a very complicated manner. In ADI, the nature of the error depends on the parameter but tends to be rather complicated. Finally, SIP has relatively smooth iteration errors.

Some of these methods produce errors that are smooth functions of the spatial coordinates. Let us consider one of these methods. The iteration error ϵ^n and residual ρ^n after the nth iteration are related by Eq. (5.15). In the Gauss-Seidel and SIP methods, after a few iterations, the rapidly varying components of the error have been removed and the error becomes a smooth function of the spatial coordinates. If the error is smooth, the update (an approximation to the iteration error) can be computed on a coarser grid. On a grid twice as coarse as the original one in two dimensions, iterations cost 1/4 as much; in three dimensions, the cost is 1/8 the fine grid cost. Furthermore, iterative methods converge much faster on coarser grids. Gauss-Seidel converges four times as fast on a grid twice as coarse; for SIP the ratio is less favorable but still substantial.

This suggests that much of the work can be done on a coarser grid. To do this, we need to define: the relationship between the two grids, the finite difference operator on the coarse grid, a method of smoothing (*restricting*) the residual from the fine grid to the coarse one and a method of interpolating

(*prolonging*) the update or correction from the coarse grid to the fine one; the words in parentheses are special terms that are in common use in the multigrid literature. Many choices are available for each item; they affect the behavior of the method but, within the range of good choices, the differences are not great. We shall thus present just one good choice for each item.

In a finite difference scheme, the coarse grid normally consists of every second line of the fine grid. In a finite volume method, one usually takes the coarse grid CVs to be composed of 2 (in two dimensions, 4, and in three dimensions, 8) fine grid CVs; the coarse grid nodes then lie between the fine grid nodes.

Although there is no reason to use the multigrid method in one dimension (because the TDMA algorithm is very effective), it is easy to illustrate the principles of the multigrid method and to derive some of the procedures used in the general case. Thus consider the problem:

$$\frac{\mathrm{d}^2\phi}{\mathrm{d}x^2} = f(x) \tag{5.60}$$

for which the standard FD approximation on a uniform grid is:

$$\frac{1}{(\Delta x)^2}\left(\phi_{i-1} - 2\phi_i + \phi_{i+1}\right) = f_i \ . \tag{5.61}$$

After performing n iterations on the grid with Δx spacing, we obtain an approximate solution ϕ^n, and the above equation is satisfied to within the residual ρ^n:

$$\frac{1}{(\Delta x)^2}\left(\phi^n_{i-1} - 2\phi^n_i + \phi^n_{i+1}\right) = f_i - \rho^n_i \ . \tag{5.62}$$

Subtracting this equation from Eq. (5.61) gives

$$\frac{1}{(\Delta x)^2}\left(\epsilon^n_{i-1} - 2\epsilon^n_i + \epsilon^n_{i+1}\right) = \rho^n_i \ , \tag{5.63}$$

which is Eq. (5.15) for node i. This is the equation we want to iterate on the coarse grid.
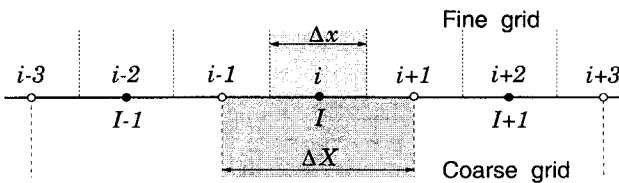
Fig. 5.2. The grids used in the multigrid technique in one dimension

To derive the discretized equations on the coarser grid, we note that control volume around node I of the coarse grid consists of the whole control volume around node i plus half of control volumes $i-1$ and $i+1$ of the fine grid (see Fig. 5.2). This suggests that we add one half of equation (5.63) with indices $i-1$ and $i+1$ to the full equation with index i, which leads to (superscript n being omitted):

$$\frac{1}{4\,(\Delta x)^2}\left(\epsilon_{i-2} - 2\epsilon_i + \epsilon_{i+2}\right) = \frac{1}{4}\left(\rho_{i-1} + 2\rho_i + \rho_{i+1}\right) . \qquad (5.64)$$

Using the relationship between the two grids ($\Delta X = 2\,\Delta x$, see Fig. 5.2), this is equivalent to the following equation on the coarse grid:

$$\frac{1}{(\Delta X)^2}\left(\epsilon_{I-1} - 2\epsilon_I + \epsilon_{I+1}\right) = \overline{\rho}_I , \qquad (5.65)$$

which also serves to define $\overline{\rho}_I$. The left hand side of this equation is the standard approximation to the second derivative on the coarse grid, indicating that the obvious discretization on the coarse grid is a reasonable one. The right hand side is a smoothing or filtering of the fine grid forcing term and provides the natural definition of the smoothing or restriction operation.

The simplest prolongation or interpolation of a quantity from the coarse grid to the fine grid is linear interpolation. At coincident points of the two grids, the value at the coarse grid point is simply *injected* onto the corresponding fine grid point. At fine grid points that lie between the coarse grid points, the injected value is the average of the neighboring coarse grid values.

A two-grid iterative method is thus:

- On the fine grid, perform iterations with a method that gives a smooth error;
- Compute the residual on the fine grid;
- Restrict the residual to the coarse grid;
- Perform iterations of the correction equation on the coarse grid;
- Interpolate the correction to the fine grid;
- Update the solution on the fine grid;
- Repeat the entire procedure until the residual is reduced to the desired level.

It is natural to ask: why not use still coarser grids to further improve the rate of convergence? This is a good idea. In fact, one should continue the procedure until it becomes impossible to define a still coarser grid; on the coarsest grid, the number of unknowns is so small that the equations can be solved exactly at a negligible cost.

Multigrid is more a strategy than a particular method. Within the framework just described there are many parameters that can be selected more or less arbitrarily: the coarse grid structure, the smoother, the number of iterations on each grid, the order in which the various grids are visited, and

the restriction and interpolation schemes are the most important of these. The rate of convergence does, of course, depend on the choices made but the range of performance between the worst and the best methods is probably less than a factor of two.

The most important property of the multigrid method is that the number of iterations on the finest grid required to reach a given level of convergence is roughly independent of the number of grid nodes. This is as good as one can expect to do – the computational cost is proportional to the number of grid nodes. In two- and three-dimensional problems with about 100 nodes in each direction, the multigrid method may converge in one-tenth to one-hundredth of the time required by the basic method. An example will be presented below.

The iterative method on which the multigrid method is based must be a good smoother; its convergence properties as a stand-alone method are less important. Gauss-Seidel and SIP are two good choices but there are other possibilities.

In two dimensions, there are many possibilities for the restriction operator. If the method described above is used in each direction, the result would be a nine point scheme. A simpler, but nearly as effective, restriction is the five point scheme:

$$\overline{\rho}_{I,J} = \frac{1}{8}\left(\rho_{i+1,j} + \rho_{i-1,j} + \rho_{i,j+1} + \rho_{i,j-1} + 4\,\rho_{i,j}\right) . \tag{5.66}$$

Similarly, an effective prolongator is bilinear interpolation. In two dimensions, there are three kinds of points on the fine grid. Those which correspond to coarse grid points are given the value at the corresponding point. Ones which lie on lines connecting two coarse grid points receive the average of the two coarse grid values. Finally, the points at the centers of coarse grid volumes take the average of the four neighbor values. Similar schemes can be derived for FV methods and 3D problems.

The initial guess in an iterative solution method is usually far from the converged solution (a zero field is often used). It therefore makes sense to solve the equation first on a very coarse grid (which is cheap) and use that solution to provide a better guess for the initial field on the next finer grid. By the time we reach the finest grid, we already have a fairly good starting solution. Multigrid methods of this type are called *full multigrid* (FMG) methods. The cost of obtaining the initial solution for the finest grid is usually more than compensated by the savings on fine grid iterations.

Finally, we remark that it is possible to construct a method in which one solves equations for approximations to the solution rather than for corrections at each grid. This is called the *full approximation scheme* (FAS) and is often used for solving non-linear problems. It is important to note that the solution obtained on each grid in FAS is *not* the solution that would be obtained if that grid were used by itself but a smoothed version of the fine grid solution;

this is achieved by passing a correction from each grid to the next coarser grid. One variant of the FAS scheme for the Navier-Stokes equations will be presented in Chap. 11.

For a detailed analysis of multigrid methods, see books by Hackbusch (1985) and Brandt (1984). A 2D multigrid solver that uses the Gauss-Seidel, SIP, or ICCG methods as the smoother is available via Internet; see the appendix for details.

5.3.9 Other Iterative Solvers

There are many other iterative solvers that cannot be described in detail here. We mention the 'red–black' variation of the Gauss-Seidel solver which is often used in conjunction with multigrid methods. On a structured grid, the nodes are imagined to be 'colored' in the same way as a checkerboard. The method consists of two Jacobi steps: black nodes are updated first, then the red nodes. When the values at black nodes are updated, only the 'old' red values are used, see Eq. (5.33). On the next step, red values are recalculated using the updated black values. This alternate application of the Jacobi method to the two sets of nodes gives an overall method with the same convergence properties as the Gauss-Seidel method. The nice feature of the red-black Gauss-Seidel solver is that it both vectorizes and parallelizes well, since there are no data dependencies in either step.

Another practice often applied to multi-dimensional problems is the use of iteration matrices which correspond to lower-dimensional problems. One version of this is the ADI method described above which reduces a 2D problem to a sequence of 1D problems. The resulting tridiagonal problems are solved line-by-line. The direction of solution is changed from iteration to iteration to improve the rate of convergence. This method is usually used in the Gauss-Seidel fashion, i.e. new variable values from lines already visited are used.

A counterpart of the red–black Gauss-Seidel method is the 'zebra' line–by–line solver: first the solution is found on the even numbered lines, then the odd numbered lines are treated. This gives better parallelization and vectorization possibilities with no sacrifice in convergence properties.

It is also possible to use the two-dimensional SIP method to solve three-dimensional problems, applying it plane–by–plane and relegating the contributions from neighboring planes to the right hand side of the equations. However, this method is neither cheaper nor faster than the three-dimensional version of SIP, so it is not often used.

5.4 Coupled Equations and Their Solution

Most problems in fluid dynamics and heat transfer require solution of coupled systems of equations, i.e. the dominant variable of each equation occurs

in some of the other equations. There are two types of approaches to such problems. In the first, all variables are solved for simultaneously. In the other, each equation is solved for its dominant variable, treating the other variables as known, and one iterates through the equations until the solution of the coupled system is obtained. The two approaches also may be mixed. We call these simultaneous and sequential methods, respectively, and they are described in more detail below.

5.4.1 Simultaneous Solution

In simultaneous methods, all the equations are considered part of a single system. The discretized equations of fluid mechanics have, after linearization, a block-banded structure. Direct solution of these equations would be very expensive, especially when the equations are non-linear and the problem is three-dimensional. Iterative solution techniques for coupled systems are generalizations of methods for single equations. The methods described above were chosen for their applicability to coupled systems. Simultaneous solution methods based on iterative solvers have been developed by several authors; see e.g. papers by Galpin and Raithby (1986), Deng et al. (1994), and Weiss et al. (1999).

5.4.2 Sequential Solution

When the equations are linear and tightly coupled, the simultaneous approach is best. However, the equations may be so complex and non-linear that coupled methods are difficult and expensive to use. It may then be preferable to treat each equation as if it has only a single unknown, temporarily treating the other variables as known, using the best currently available values for them. The equations are then solved in turn, repeating the cycle until all equations are satisfied. When using this type of method, two points need to be borne in mind:

- Since some terms, e.g. the coefficients and source terms that depend on the other variables change as the computation proceeds, it is inefficient to solve the equations accurately at each iteration. That being the case, direct solvers are unnecessary and iterative solvers are preferred. Iterations performed on each equation are called *inner* iterations.
- In order to obtain a solution which satisfies all of the equations, the coefficient matrices and source vector must be updated after each cycle and the process repeated. The cycles are called *outer* iterations.

Optimization of this type of solution method requires careful choice of the number of inner iterations per outer iteration. It is also necessary to limit the change in each variable from one outer iteration to the next (under-relaxation), because a change in one variable changes the coefficients in the

other equations, which may slow or prevent convergence. Unfortunately, it is hard to analyze the convergence of these methods so the selection of under-relaxation factors is largely empirical.

The multigrid method, which was described above as a convergence accelerator for inner iterations (linear problems), may be applied to coupled problems. It may also be used to accelerate the outer iterations as will be described in Chap. 11.

5.4.3 Under-Relaxation

We shall present one under-relaxation technique that is widely used. On the nth outer iteration, the algebraic equation for a generic variable, ϕ, at a typical point P may be written:

$$A_{\mathrm{P}}\phi_{\mathrm{P}}^{n} + \sum_{l} A_{l}\phi_{l}^{n} = Q_{\mathrm{P}} , \tag{5.67}$$

where Q contains all the terms that do not depend explicitly on ϕ^n; the coefficients A_l and the source Q may involve ϕ^{n-1}. The discretization scheme is unimportant here. This equation is linear and the system of equations for the whole solution domain is solved usually iteratively (inner iterations).

In the early outer iterations, allowing ϕ to change by as much as Eq. (5.67) requires could cause instability, so we allow ϕ^n to change only a fraction α_ϕ of the would-be difference:

$$\phi^{n} = \phi^{n-1} + \alpha_{\phi}(\phi^{\mathrm{new}} - \phi^{n-1}) , \tag{5.68}$$

where ϕ^{new} is the result of Eq. (5.67) and the under-relaxation factor satisfies $0 < \alpha_\phi < 1$.

Since the old iterate is usually no longer required after the coefficient matrix and source vector are updated, the new solution can be written over it. Replacing ϕ^{new} in Eq. (5.68) by

$$\phi_{\mathrm{P}}^{\mathrm{new}} = \frac{Q_{\mathrm{P}} - \sum_{l} A_{l}\phi_{l}^{n}}{A_{\mathrm{P}}} , \tag{5.69}$$

which follows from Eq. (5.67), leads to a modified equation at node P:

$$\underbrace{\frac{A_{\mathrm{P}}}{\alpha_{\phi}}}_{A_{\mathrm{P}}^{*}} \phi_{\mathrm{P}}^{n} + \sum_{l} A_{l}\phi_{l}^{n} = \underbrace{Q_{\mathrm{P}} + \frac{1-\alpha_{\phi}}{\alpha_{\phi}} A_{\mathrm{P}}\phi_{\mathrm{P}}^{n-1}}_{Q_{\mathrm{P}}^{*}} , \tag{5.70}$$

where A_{P}^{*} and Q_{P}^{*} are modified main diagonal matrix elements and source vector components. This modified equation is solved within inner iterations. When the outer iterations converge, the terms involving α_ϕ cancel out and we obtain the solution of the original problem.

This kind of under-relaxation was proposed by Patankar (1980). It has a positive effect on many iterative solution methods since the diagonal dominance of the matrix A is increased (the element A_P^* is larger than A_P, while A_l remains the same). It is more efficient than explicit application of the expression (5.68).

Optimum under-relaxation factors are problem dependent. A good strategy is to use a small under-relaxation factor in the early iterations and increase it towards unity as convergence is approached. Some guidance for selecting the under-relaxation factors for solving the Navier-Stokes equations will be given in Chaps. 7 and 8. Under-relaxation may be applied not only to the dependent variables but also to individual terms in the equations. It is often necessary to do so when the fluid properties (viscosity, density, Prandtl number etc.) depend on the solution and need be updated.

We mentioned above that iterative solution methods can often be regarded as solving an unsteady problem until a steady state is reached. Control of the time step is then important in controlling the evolution of the solution. In the next chapter we shall show that time step may be interpreted as an under-relaxation factor. The under-relaxation scheme described above may be interpreted as using different time steps at different nodes.

5.5 Non-Linear Equations and their Solution

As mentioned above, there are two types of techniques for solving non-linear equations: Newton-like and global. The former are much faster when a good estimate of the solution is available but the latter are guaranteed not to diverge; there is a trade-off between speed and security. Combinations of the two methods are often used. There is a vast literature devoted to methods for solving non-linear equations but the state-of-the-art is still not completely satisfactory. We cannot cover even a substantial fraction of the methods here and give a short overview of some methods.

5.5.1 Newton-like Techniques

The master method for solving non-linear equations is Newton's method. Suppose that one needs to find the root of a single algebraic equation $f(x) = 0$. Newton's method linearizes the function about an estimated value of x using the first two terms of the Taylor series:

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) \,. \tag{5.71}$$

Setting the linearized function equal to zero provides a new estimate of the root:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \quad \text{or, in general,} \quad x_k = x_{k-1} - \frac{f(x_{k-1})}{f'(x_{k-1})} \tag{5.72}$$

and we continue until the change in the root $x_k - x_{k-1}$ is as small as desired. The method is equivalent to approximating the curve representing the function by its tangent at x_k. When the estimate is close enough to the root, this method converges quadratically i.e. the error at iteration $k+1$ is proportional to the square of the error at iteration k. This means that only a few iterations are needed once the solution estimate is close to the root. For that reason, it is employed whenever it is feasible to do so.

Newton's method is easily generalized to systems of equations. A generic system of non-linear equations can be written:

$$f_i(x_1, x_2, \ldots, x_n) = 0 , \quad i = 1, 2, \ldots, n . \tag{5.73}$$

This system can be linearized in exactly the same way as the single equation. The only difference is that now we need to use multi-variable Taylor series:

$$f_i(x_1, x_2, \ldots, x_n) = f_i(x_1^k, x_2^k, \ldots, x_n^k) + \sum_{j=1}^{n} (x_j^{k+1} - x_j^k) \frac{\partial f_i(x_1^k, x_2^k, \ldots, x_n^k)}{\partial x_j} , \tag{5.74}$$

for $i = 1, 2, \ldots, n$. When this is set to zero, we have a system of linear algebraic equations that can be solved by Gauss elimination or some other technique. The matrix of the system is the set of partial derivatives:

$$a_{ij} = \frac{\partial f_i(x_1^k, x_2^k, \ldots, x_n^k)}{\partial x_j} , \quad i = 1, 2, \ldots, n , \quad j = 1, 2, \ldots, n , \tag{5.75}$$

which is called the Jacobian of the system. The system of equations is:

$$\sum_{j=1}^{n} a_{ij} (x_j^{k+1} - x_j^k) = -f_i(x_1^k, x_2^k, \ldots, x_n^k) , \quad i = 1, 2, \ldots, n . \tag{5.76}$$

For an estimate that is close to the correct root, Newton's method for systems converges as rapidly as the method for a single equation. However, for large systems, the rapid convergence is more than offset by its principal disadvantage. For the method to be effective, the Jacobian has to be evaluated at each iteration. This presents two difficulties. The first is that, in the general case, there are n^2 elements of the Jacobian and their evaluation becomes the most expensive part of the method. The second is that a direct method of evaluating the Jacobian may not exist; many systems are such that the equations are implicit or they may be so complicated that differentiation is all but impossible.

To the authors' knowledge, Newton's method has been used only few times to solve the Navier-Stokes equations although it has been used to solve simplifications of these equations many times. It was found that the cost of generating the Jacobian and solving the system by Gauss elimination was so

high that, even though the method does converge in just a few iterations, the overall cost is greater than that of other iterative methods.

For generic systems of non-linear equations, secant methods are much more effective. For a single equation, the secant method approximates the derivative of the function by the secant drawn between two points on the curve. This method converges more slowly than Newton's method, but as it does not require evaluation of the derivative, it may find the solution at lower overall cost and can be applied to problems in which direct evaluation of the derivative is not possible. There are a number of generalizations of the secant method to systems, most of which are quite effective but, as they have not been applied in CFD, we shall not review them here.

5.5.2 Other Techniques

The usual approach to the solution of coupled non-linear equations is the sequential decoupled method described in the previous section. The non-linear terms (convective flux, source term) are usually linearized using *Picard iteration* approach. For the convective terms, this means that the mass flux is treated as known, so the non-linear convective term in the equation for the u_i momentum component is approximated by:

$$\rho u_j u_i \approx (\rho u_j)^o u_i \ , \tag{5.77}$$

where index o denotes that the values are taken from the result of the previous outer iteration. Similarly, the source term is decomposed into two parts:

$$q_\phi = b_0 + b_1 \phi \ . \tag{5.78}$$

The portion b_0 is absorbed into the right hand side of the algebraic equation, while b_1 contributes to the coefficient matrix A. A similar approach can be used for the non-linear terms that involve more than one variable.

This kind of linearization requires many more iterations than a coupled technique using Newton-like linearization. However, the number of outer iterations can be reduced using multigrid techniques, which makes this approach attractive.

Newton's method is sometimes used to linearize the non-linear terms; for example, the convective term in the equation for the u_i momentum component can be expressed as:

$$\rho u_j u_i \approx \rho u_j^o u_i + \rho u_i^o u_j - \rho u_j^o u_i^o \ . \tag{5.79}$$

Non-linear source terms can be treated in the same way. This leads to a coupled linear system of equations which is difficult to solve, and the convergence is not quadratic unless the full Newton technique is used. However, special coupled iterative techniques which benefit from this linearization technique can be developed, as shown by Galpin and Raithby (1986).

5.6 Deferred-Correction Approaches

If all terms containing the nodal values of the unknown variable are kept on the left-hand side of Eq. (3.42), the computational molecule may become very large. Since the size of the computational molecule affects both the storage requirements and the effort needed to solve the linear equation system, we would like to keep it as small as possible; usually, only the nearest neighbors of node P are kept on the left hand sides of the equations. However, approximations which produce such simple computational molecules are usually not accurate enough, so we are forced to use approximations that refer to more nodes than just the nearest neighbors.

One way around this problem would be to leave only the terms containing the nearest neighbors on the left-hand side of Eq. (3.42) and bring all other terms to the right-hand side. This requires that these terms be evaluated using values from the previous iteration. However, this is not a good practice and may lead to the divergence of the iterations because the terms treated explicitly may be substantial. To prevent divergence, strong under-relaxation of the changes from one iteration to the next would be required (see Sect. 5.4.3), resulting in slow convergence.

A better approach is to compute the terms that are approximated with a higher-order approximation explicitly and put them on the right-hand side of the equation. Then one takes a simpler approximation to these terms (one that gives a small computational molecule) and puts it both on the left-hand side of the equation (with unknown variable values) and on the right-hand side (computing it explicitly using existing values). The right-hand side is now the difference between two approximations of the same term, and is likely to be small. Thus it should cause no problems in the iterative solution procedure. Once the iterations converge, the low order approximation terms drop out and the obtained solution corresponds to the higher-order approximation.

Since iterative methods are usually necessary due to the non-linearity of the equations to be solved, adding a small term to the part treated explicitly increases the computing effort by only a small amount. On the other hand, both the memory and computing time required are greatly reduced when the size of the computational molecule in the part of the equation treated implicitly is small.

We shall refer to this technique very often. It is used when treating higher-order approximations, grid non-orthogonality, and corrections needed to avoid undesired effects like oscillations in the solution. Because the right-hand side of the equation can be regarded as a "correction" this method is called deferred correction. Here its use will be demonstrated in conjunction with Padé schemes in FD (see Sect. 3.3.3) and with higher-order interpolations in FV-methods (see Sect. 4.4.4).

If Padé schemes are to be used in implicit FD-methods, deferred correction must be employed since approximation of the derivative at one node involves derivatives at neighboring nodes. One approach is to use the "old values"

of the derivatives at neighboring nodes and variable values at distant nodes. These are usually taken from the result of the preceding iteration and we have:

$$\left(\frac{\partial \phi}{\partial x}\right)_i = \beta \frac{\phi_{i+1} - \phi_{i-1}}{2\,\Delta x} + \gamma \left(\frac{\phi_{i+2} - \phi_{i-2}}{4\,\Delta x}\right)^{\text{old}} - \alpha \left(\frac{\partial \phi}{\partial x}\right)_{i+1}^{\text{old}} - \alpha \left(\frac{\partial \phi}{\partial x}\right)_{i-1}^{\text{old}} \tag{5.80}$$

In this case, only the first term on the right hand side of this equation will be moved to the left hand side of the equation to be solved at the new outer iteration.

However, this approach may affect the convergence rate adversely since the implicitly treated part is not an approximation to the derivative but, rather, some multiple of it. The following version of deferred correction is more efficient:

$$\left(\frac{\partial \phi}{\partial x}\right)_i = \frac{\phi_{i+1} - \phi_{i-1}}{2\,\Delta x} + \left[\left(\frac{\partial \phi}{\partial x}\right)_i^{\text{Padé}} - \frac{\phi_{i+1} - \phi_{i-1}}{2\,\Delta x}\right]^{\text{old}} . \tag{5.81}$$

Here, the complete second-order CDS approximation is used on the left hand side. On the right-hand side we have the difference between the explicitly computed Padé scheme derivative and the explicitly computed CDS-approximation. This gives a more balanced expression because, where the second-order CDS is accurate enough, the term in square brackets is negligible. Instead of CDS, we could use UDS for the implicit part; in that case, the UDS approximation should be used on both sides of the equation.

Deferred correction is also useful in FV-methods when higher-order schemes are used (see Sect. 4.4.4). Higher-order flux approximations are computed *explicitly* and this approximation is then combined with an implicit lower-order approximation (which uses only variable values at nearest neighbors) in the following way (first suggested by Khosla and Rubin, 1974):

$$F_{\text{e}} = F_{\text{e}}^{\text{L}} + \left(F_{\text{e}}^{\text{H}} - F_{\text{e}}^{\text{L}}\right)^{\text{old}} . \tag{5.82}$$

F_{e}^{L} stands for the approximation by some lower-order scheme (UDS is often used for convective and CDS for diffusive fluxes) and F_{e}^{H} is the higher-order approximation. The term in brackets is evaluated using values from the previous iteration, as indicated by the superscript 'old'. It is normally small compared to the implicit part, so its explicit treatment does not affect the convergence significantly.

The same approach can be applied to all high-order approximations including spectral methods. Although deferred correction increases the computation time per iteration relative to that for a pure low-order scheme, the

additional effort is much smaller than that needed to treat the entire higher-order approximation implicitly. One can also multiply the "old" term with a blending factor between zero and unity to produce a mixture of the pure low-order and pure high-order schemes. This is sometimes used to avoid oscillations which result when higher-order schemes are used on grids that are not sufficiently fine. For example, when flow around a body is computed, one would like to use a fine grid near the body and a coarser grid far from it. A high-order scheme may produce oscillations in the coarse-grid region, thus spoiling the whole solution. Since the variables vary slowly in the coarse-grid region, we may reduce the order of approximation there without affecting the solution in the fine-grid region; this can be achieved by using a blending factor in the outer region only.

More details on other uses of deferred-correction approach will be given in subsequent chapters.

5.7 Convergence Criteria and Iteration Errors

When using iterative solvers, it is important to know when to quit. The most common procedure is based on the difference between two successive iterates; the procedure is stopped when this difference, measured by some norm, is less than a pre-selected value. Unfortunately, this difference may be small when the error is not small and a proper normalization is essential.

From the the analysis presented in Sect. 5.3.2, we find (see Eqs. (5.14) and (5.27)):

$$\boldsymbol{\delta}^n = \boldsymbol{\phi}^{n+1} - \boldsymbol{\phi}^n \approx (\lambda_1 - 1)(\lambda_1)^n a_1 \boldsymbol{\psi}_1 \ , \tag{5.83}$$

where δ^n is the difference between solution at iterations $n+1$ and n, and λ_1 is the largest eigenvalue or spectral radius of the iteration matrix. It can be estimated from:

$$\lambda_1 \approx \frac{||\boldsymbol{\delta}^n||}{||\boldsymbol{\delta}^{n-1}||} \ , \tag{5.84}$$

where $||\boldsymbol{a}||$ represents the norm (e.g. root mean square or L_2 norm) of $\boldsymbol{a}$.

Once an estimate of the eigenvalue is available, it is not difficult to estimate the iteration error. In fact, by rearranging Eq. (5.83), we find:

$$\boldsymbol{\epsilon}^n = \boldsymbol{\phi} - \boldsymbol{\phi}^n \approx \frac{\boldsymbol{\delta}^n}{\lambda_1 - 1} \ . \tag{5.85}$$

A good estimate of the iteration error is therefore:

$$||\boldsymbol{\epsilon}^n|| \approx \frac{||\boldsymbol{\delta}^n||}{\lambda_1 - 1} \tag{5.86}$$

This error estimate can be computed from the two successive iterates of the solution. Although the method is designed for linear systems, all systems are essentially linear near convergence; as this is the time when error estimates are most needed, the method can be applied to nonlinear systems as well.

Unfortunately, iterative methods often have complex eigenvalues. When this is the case, the error reduction is not exponential and may not be monotonic. Since the equations are real, complex eigenvalues must occur as conjugate pairs. Their estimation requires an extension of the above procedure. In particular, data from more iterates are required. Some of the ideas used below are found in Golub and van Loan (1990).

If the eigenvalues of largest magnitude are complex, there are at least two of them and Eq. (5.27) must be replaced by

$$\boldsymbol{\epsilon}^n \approx a_1(\lambda_1)^n \boldsymbol{\psi}_1 + a_1^*(\lambda_1^*)^n \boldsymbol{\psi}_1^* \,, \tag{5.87}$$

where * indicates the conjugate of a complex quantity. As before, we subtract two successive iterates to obtain $\boldsymbol{\delta}^n$, see Eq. (5.83). If we further let:

$$\boldsymbol{\omega} = (\lambda_1 - 1) a_1 \boldsymbol{\psi}_1 \,, \tag{5.88}$$

then the following expression is obtained:

$$\boldsymbol{\delta}^n \approx (\lambda_1)^n \boldsymbol{\omega} + (\lambda_1^*)^n \boldsymbol{\omega}^* \,. \tag{5.89}$$

Since the magnitude of the eigenvalue λ_1 is the quantity of greatest interest, we write:

$$\lambda_1 = \ell\, \mathrm{e}^{\mathrm{i}\vartheta} \,. \tag{5.90}$$

A straight-forward calculation then shows that:

$$z^n = \boldsymbol{\delta}^{n-2} \cdot \boldsymbol{\delta}^n - \boldsymbol{\delta}^{n-1} \cdot \boldsymbol{\delta}^{n-1} = 2\ell^{2n-2} |\boldsymbol{\omega}|^2 [\cos(2\vartheta) - 1] \,, \tag{5.91}$$

from which it is easy to show that:

$$\ell = \sqrt{\frac{z^n}{z^{n-1}}} \tag{5.92}$$

is an estimate of the magnitude of the eigenvalue.

Estimation of the error requires further approximations. The complex eigenvalues cause the errors to oscillate and the shape of the error is not independent of the iteration number, even for large n. To estimate the error, we compute $\boldsymbol{\delta}^n$ and ℓ from expressions given above. Due to the complex eigenvalues and eigenvectors, the result contains terms proportional to the cosine of the phase angle. As we are interested only in magnitudes, we assume that these terms are zero in an average sense and drop them. This allows us to find a simple relationship between the two quantities:

$$\boldsymbol{\epsilon}^n \approx \frac{\boldsymbol{\delta}^n}{\sqrt{\ell^2 + 1}} \,. \tag{5.93}$$

This is the desired estimate of the error. Due to the oscillations in the solution, the estimate may not be accurate on any particular iteration, but, as we shall show below, it is quite good on the average.

In order to remove some of the effects of the oscillation, the eigenvalue estimates should be averaged over a range of iterations. Depending on the problem and the number of anticipated iterations, the averaging range may vary from 1 to 50 (typically 1% of the expected number of iterations).

Finally, we want a method that can treat both real and complex eigenvalues. The error estimator for the complex case (5.93) gives low estimates if the principal eigenvalue (λ_1) is real. Also, the contribution of λ_1 to z^n drops out in this case so the eigenvalue estimate is quite poor. However, this fact can be used to determine whether λ_1 is real or complex. If the ratio

$$r = \frac{z^n}{|\delta^n|^2} \tag{5.94}$$

is small, the eigenvalue is probably real; if r is large, the eigenvalue is probably complex. For real eigenvalues, r tends to be smaller than 10^{-2} and, for complex eigenvalues, $r \approx 1$. One can therefore adopt a value of $r = 0.1$ as an indicator of type of eigenvalue and use the appropriate expression for the error estimator.

A compromise is to use the reduction of the residual as a stopping criterion. Iteration is stopped when the residual norm has been reduced to some fraction of its original size (usually by three or four orders of magnitude). As we have shown, the iteration error is related to the residual via Eq. (5.15) so reduction of the residual is accompanied by reduction of the iteration error. If the iteration is started from zero initial values, than the initial error is equal to the solution itself. When the residual level has fallen say three to four orders of magnitude below the initial level, the error is likely to have fallen by a comparable amount, i.e. it is of the order of 0.1% of the solution. The residual and the error usually do not fall in the same way at the beginning of iteration process; caution is also needed because, if the matrix is poorly conditioned, the error may be large even when the residual is small.

Many iterative solvers require calculation of the residual. The above approach is then attractive as it requires no additional computation. The norm of residual prior to the first inner iteration provides a reference for checking the convergence of inner iterations. At the same time it provides a measure of the convergence of the outer iterations. Experience shows that inner iterations can be stopped when the residual has fallen by one to two orders of magnitude. Outer iterations should not be stopped before the residual has been reduced by three to five orders of magnitude, depending on the desired accuracy. The sum of absolute residuals (the L_1 norm) can be used instead of the r.m.s. (L_2) norm. The convergence criterion should be more stringent on refined grids, because the discretization errors are smaller on them than on coarse grids.

If the order of the initial error is known, it is possible to monitor the norm of the difference between two iterates and compare it with the same quantity at the beginning of the iteration process. When the difference norm has fallen three to four orders of magnitude, the error has usually fallen by a comparable amount.

Both of these methods are only approximate; however, they are better than the criterion based on the non-normalized difference between two successive iterates.
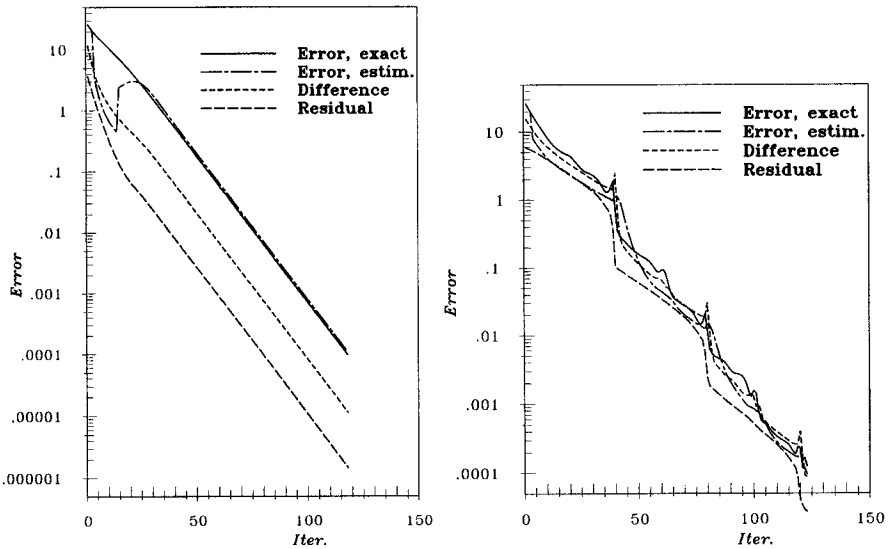
Fig. 5.3. Variation of the norm of the exact iteration error, error estimate, residual and difference between two iterations for the Laplace problem with the SOR solver on a 20 × 20 CV grid: relaxation parameter smaller (left) and larger (right) than the optimum

In order to test the method of estimating iteration errors, we first present the solution of a linear 2D problem using the SOR solver. The linear problem is Laplace equation in the square domain $\{0 < x < 1;\ 0 < y < 1\}$ with Dirichlet boundary conditions chosen to correspond to the solution $\phi(x, y) = 100\,xy$. The advantage of this choice is that the second order central difference approximation to the converged solution is exact on any grid so the actual difference between the present iterate and the converged solution is easily computed. The initial guess of the solution is zero everywhere within the domain. We chose the SOR method as the iterative technique because, if the relaxation parameter is greater than the optimum value, the eigenvalues are complex.

Results are shown for uniform grids with 20×20 and 80×80 CV in Figs. 5.3 and 5.4. In each case, the norms of the exact iteration error, the error estimate using the above described technique, the difference between two iterates and the residual are shown. For both cases the results of calculation for two values of the relaxation parameter are shown: one below the optimum value, which has real eigenvalues, and one above the optimum, leading to complex eigenvalues. For the case of real eigenvalues, smooth exponential convergence results. The error estimate is almost exact in this case (except in the initial period). However, the norms of the residual and difference between two iterates initially fall too rapidly and do not follow the fall of the iteration error. This effect is more pronounced as the grid is refined. On the 80×80 CV grid, the residual norm is quickly reduced by two orders of magnitude while the error is only slightly reduced. Once the asymptotic reduction rate is achieved, the slopes of all four curves are the same.
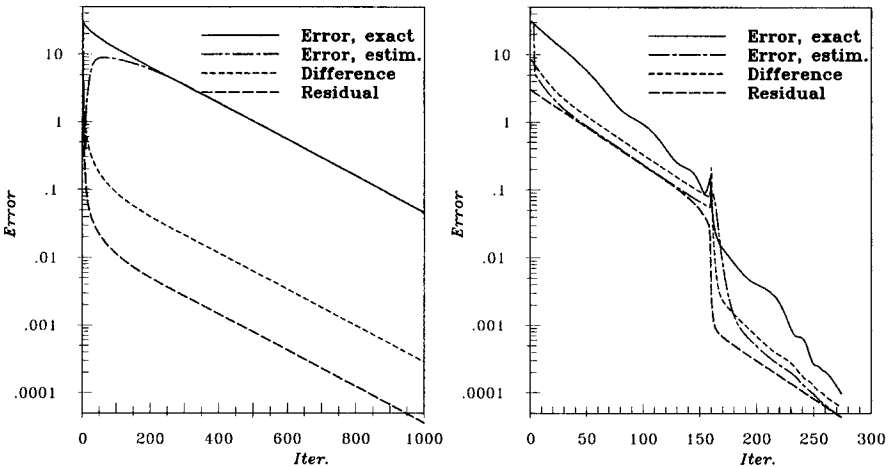
Fig. 5.4. Variation of the norm of the exact iteration error, error estimate, residual and difference between two iterations for the Laplace problem with the SOR solver on a 80×80 CV grid: relaxation parameter smaller (left) and larger (right) than the optimum

In the case for which the eigenvalues of the iteration matrix are complex, the convergence is not monotonic – there are oscillations in the error. The comparison of predicted and exact errors is in this case also quite satisfactory. All of the above mentioned convergence criteria seem to be equally good in this case.

Further examples of the estimation of iteration errors, especially for the outer iterations in case of solving coupled flow problems, will be presented later.

5.8 Examples

In the previous chapter we presented solutions of some 2D problems without discussing the methods of solution. We shall now show the performance of various solvers for the case of scalar transport in a stagnation point flow; see Sect. 4.7 for the description of problem and discretization techniques used to derive the linear algebraic equations.

We consider the case with $\Gamma = 0.01$ and uniform grids with 20×20, 40×40 and 80×80 CVs. The equation matrix A is not symmetric and, in case of CDS discretization, it is not diagonally dominant. In a diagonally dominant matrix, the element on the main diagonal satisfies the following condition:

$$A_{\rm P} \geq \sum_l |A_l| \, . \tag{5.95}$$

It can be shown that a sufficient condition for convergence of iterative solution methods is that the above relation be satisfied, and that inequality must apply at least at one node. This condition is satisfied only by UDS discretization of convective terms. While simple solvers like Jacobi and Gauss-Seidel usually diverge when the above condition is violated, ILU, SIP and conjugate gradient solvers are less sensitive to diagonal dominance of the matrix.

We considered five solvers:

- Gauss-Seidel, denoted by GS;
- Line Gauss-Seidel using TDMA on lines x = const., denoted by LGS-X;
- Line Gauss-Seidel using TDMA on lines y = const., denoted by LGS-Y;
- Line Gauss-Seidel using TDMA alternately on lines x = const. and y = const., denoted by LGS-ADI;
- Stone's ILU method, denoted by SIP.

Table 5.1 shows numbers of iterations required by the above solvers to reduce the absolute residual sum by four orders of magnitude.

Table 5.1. Numbers of iterations required by various solvers to reduce the L_1 residual norm by four orders of magnitude when solving the 2D scalar transport problem in stagnation point flow

Scheme	Grid	GS	LGS-X	LGS-Y	LGS-ADI	SIP
	20 × 20	68	40	35	18	14
UDS	40 × 40	211	114	110	52	21
	80 × 80	720	381	384	175	44
	20 × 20	-	-	-	12	19
CDS	40 × 40	163	95	77	39	19
	80 × 80	633	349	320	153	40

From the table we see that LGS-X and LGS-Y solvers are about twice as fast as GS; LGS-ADI is about twice as fast as LGS-X, and on the finer grids,

SIP is about four times as fast as LGS-ADI. For the GS and LGS solvers, the number of iterations increases by about a factor of four each time the grid is refined; the factor is smaller in case of SIP and LGS-ADI, but, as we shall see in the next example, the factor increases and asymptotically approaches four in the limit of very fine grids.

Another interesting observation is that the GS and LGS solvers do not converge on a 20×20 CV grid with CDS discretization. This is due to the fact that the matrix is not diagonally dominant in this case. Even on the 40×40 CV grid, the matrix is not completely diagonally dominant, but the violation is in the region of uniform distribution of the variable (low gradients) so the effect on the solver is not as severe. The LGS-ADI and SIP solvers are not affected.

We turn now to a test case for which an analytical solution exists and the CDS approximation produces exact solution on any grid. This helps in the evaluation of the iteration error but the behavior of the solver does not benefit so this case is quite suitable for assessing solver performance. We are solving the Laplace equation with Dirichlet boundary conditions, for which the exact solution is $\phi = xy$. The solution domain is a rectangle, the solution is prescribed at all boundaries and the initial values in the interior are all zero. The initial error is thus equal to the solution and is a smooth function of spatial coordinates. Discretization is performed using FV method described in the previous section and CDS scheme. Since the convection is absent, the problem is fully elliptic.

The solvers considered are:

- Gauss-Seidel solver, denoted by GS;
- Line Gauss-Seidel solver using TDMA alternately along lines x = const. and y = const., denoted by LGS-ADI;
- ADI solver described in Sect. 5.3.5;
- Stone's ILU method, denoted by SIP;
- Conjugate gradient method, preconditioned using incomplete Cholesky decomposition, denoted ICCG;
- Multigrid method using GS as a smoother, denoted MG-GS;
- Multigrid method using SIP as a smoother, denoted MG-SIP.

Table 5.2 shows results obtained on uniform grids on a square solution domain. LGS-ADI is again about four times as fast as GS, and the SIP is about four times as fast as LGS-ADI. ADI is less efficient than SIP on coarse grids but, when the optimum time step is chosen, the number of iterations only doubles when the number of grid points in one direction is doubled, so for fine grids it is quite effective. When the time step is varied in a cyclic fashion, the solver becomes even more efficient. This is also true of SIP, but cyclic variation of the parameter increases the cost per iteration. The number of iterations required by ADI to reach convergence for various time steps is given in Table 5.3. The optimum time step is reduced by a factor of two when the grid is refined.

Table 5.2. Numbers of iterations required by various solvers to reduce the normalized L_1 error norm below 10^{-5} for the 2D Laplace equation with Dirichlet boundary conditions on a square domain $X \times Y = 1 \times 1$ with uniform grid in both directions

Grid	GS	LGS-ADI	ADI	SIP	ICCG	MG-GS	MG-SIP
8 × 8	74	22	16	8	7	12	7
16 × 16	292	77	31	20	13	10	6
32 × 32	1160	294	64	67	23	10	6
64 × 64	4622	1160	132	254	46	10	6
128 × 128	-	-	274	1001	91	10	6
256 × 256	-	-	-	-	181	10	6

Table 5.3. Numbers of iterations required by ADI solver as a function of the time step size (uniform grid in both directions, 64 × 64 CV)

$1/\Delta t$	80	68	64	60	32	16	8
No. Iter.	152	134	132	134	234	468	936

ICCG is substantially faster than SIP; the number of iterations doubles when the grid is refined, so its advantage is bigger on fine grids. Multigrid solvers are very efficient; with SIP as the smoother, only 6 iterations on the finest grid are required. In the MG solvers, the coarsest level was 2 × 2 CV, so there were three levels on the 8 × 8 CV grid and eight levels on the 256 × 256 CV grid. One iteration was performed on the finest grid and on all grids after prolongation, while 4 iterations were performed during the restriction phase. In SIP, the parameter α was set to 0.92. No attempt was made to find optimum values of the parameters; the results obtained are representative enough to show the trends and the relative performance of the various solvers. Note also that the computing effort per iteration is different for each solver. Using the cost of a GS iteration as a reference, we find the following relative costs: LGS-ADI – 2.5, ADI – 3.0, SIP – 4.0 for the first iteration and 2.0 afterwards, ICCG – 4.5 for the first iteration and 3.0 afterwards. For MG methods, one needs to multiply the number of iterations on the finest grid by roughly 1.5 to account for the cost of iterations on coarse grids. MG-GS is therefore computationally the most efficient solver in this case.

Since the rate of convergence is different for each solver, the relative cost depends on how accurately we want to solve the equations. To analyze this issue we have plotted the variation of the sum of absolute residuals, and the variation of iteration error with iterations in Fig. 5.5. Two observations can be made:

- The fall of the residual sum is irregular initially, but after a certain number of iterations, the convergence rate becomes constant. An exception is the ICCG solver, which becomes faster as iterations go on. When very accurate solution is needed, MG solvers and ICCG are the best choice. If – as is the
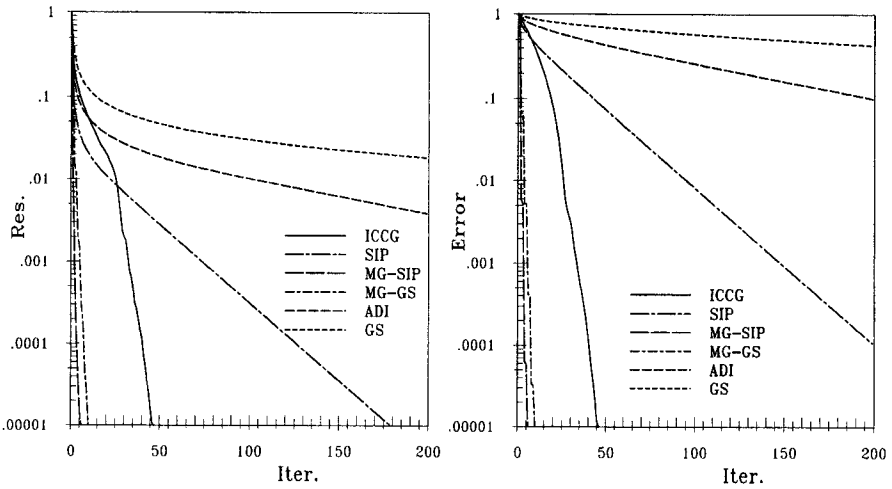
Fig. 5.5. Variation of the L_1 norm of residual (left) and iteration error (right) as a function of the number of performed iterations for various solvers and a 64 × 64 CV grid

case when solving non-linear problems – moderate accuracy is needed, SIP becomes competitive, and even ADI may be good enough in this case.

- The initial reduction of residual norm is not accompanied by an equal reduction of iteration error for the GS, SIP, ICCG and ADI solvers. Only MG solvers reduce the error and the residual norm at the same pace.

These conclusions are quite general, although there are problem dependent features. We shall show similar results for the Navier-Stokes equations later.
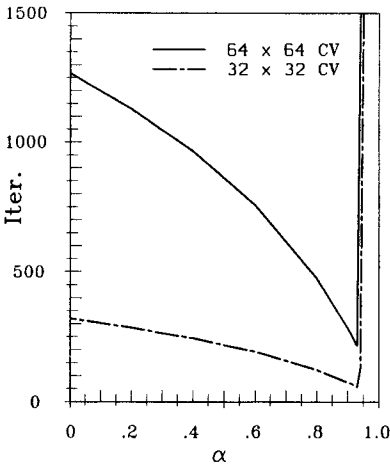
Fig. 5.6. Number of iterations required to reduce the L_1 residual norm below 10^{-4} in the above 2D Laplace problem using SIP solver, as a function of the parameter α

Since the SIP solver is used in CFD on structured grids, we show the dependence of the number of iterations required to reach convergence on the parameter α in Fig. 5.6. For $\alpha = 0$ the SIP reduces to the standard ILU solver. With the optimum value of α, SIP is about six times as fast as ILU. The problem with SIP is that the optimum value of α lies at the end of the range of usable values: for α slightly larger than the optimum value, the method does not converge. The optimum value usually lies between 0.92 and 0.96. It is safe to use $\alpha = 0.92$, which is usually not optimum, but it provides about five times the speed of the standard ILU method.

Some solvers are affected by cell aspect ratio, because the magnitudes of coefficients becomes non-uniform. Using a grid with $\Delta x = 10\,\Delta y$ makes $A_{\rm N}$ and $A_{\rm S}$ 100 times larger than $A_{\rm W}$ and $A_{\rm E}$ (see example section of the previous chapter). To investigate this effect we solved the Laplace equation problem described above on a rectangular region $X \times Y = 10 \times 1$ using the same number of grid nodes in each direction. Table 5.4 shows numbers of iterations required to reduce the normalized L_1 residual norm below 10^{-5} for various solvers. The GS solver is not affected, but it is no longer a suitable smoother for the MG method. LGS-ADI and SIP solvers become substantially faster compared to the square grid problem. ICCG also performs slightly better. MG-SIP is not affected but MG-GS deteriorates considerably.

Table 5.4. Numbers of iterations required by various solvers to reduce the normalized L_1 residual norm below 10^{-5} for the 2D Laplace equation with Dirichlet boundary conditions on a rectangular domain $X \times Y = 10 \times 1$ with uniform grid in both directions

Grid	GS	LGS-ADI	SIP	ICCG	MG-GS	MG-SIP
8 × 8	74	5	4	4	54	3
16 × 16	293	8	6	6	140	4
32 × 32	1164	18	13	11	242	5
64 × 64	4639	53	38	21	288	6
128 × 128	-	189	139	41	283	6
256 × 256	-	-	-	82	270	6

This behavior is typical and is also found in convection/diffusion problems (although the effect is less pronounced) and on non-uniform grids which have both small and large cell aspect ratios. A mathematical explanation for the worsening of GS and improvement of ILU performance with increasing aspect ratio is given by Brandt (1984).

Finally we present some results for the solution of a Poisson equation with Neumann boundary conditions in 3D. The pressure and pressure-correction equations in CFD are of this type. The equation solved is:

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} + \frac{\partial^2 \phi}{\partial z^2} = \sin(x^* \pi)\ \sin(y^* \pi)\ \sin(z^* \pi)\ , \tag{5.96}$$

Table 5.5. Numbers of iterations required by various solvers to reduce the L_1 residual norm below 10^{-4} for the 3D Poisson equation with Neumann boundary conditions

Grid	GS	SIP	ICCG	CGSTAB	FMG-GS	FMG-SIP
8^3	66	27	10	7	10	6
16^3	230	81	19	12	10	6
32^3	882	316	34	21	9	6
64^3	-	1288	54	41	7	6

where $x^* = x/X$, $y^* = y/Y$, $z^* = z/Z$, and X, Y, Z are the dimensions of the solution domain. The equation is discretized using FV method. The sum of the source terms over the domain is zero, and Neumann boundary conditions (zero gradient normal to boundary) were specified at all boundaries. In addition to GS, SIP and ICCG solvers introduced above, we used also the CGSTAB method with incomplete Cholesky preconditioning. The initial solution is zero. The numbers of iterations required to reduce the normalized sum of absolute residuals four orders of magnitude are presented in Table 5.5.

The conclusions reached from this exercise are similar to those drawn from 2D problems with Dirichlet boundary conditions. When an accurate solution is required, GS and SIP become inefficient on fine grids; conjugate gradient solvers are a better choice, and multigrid methods are best. The FMG strategy, in which the solution on a coarse grid provides the initial solution for the next finer grid, is better than straight multigrid. FMG with ICCG or CGSTAB as a smoother requires even fewer iterations (three to four on the finest grid), but the computing time is higher than for MG-SIP. The FMG principle can be applied to other solvers as well.

6. Methods for Unsteady Problems

6.1 Introduction

In computing unsteady flows, we have a fourth coordinate direction to consider: *time*. Just as with the space coordinates, time must be discretized. We can consider the time "grid" in either the finite difference spirit, as discrete points in time, or in a finite volume view as "time volumes". The major difference between the space and time coordinates lies in the direction of influence: whereas a force at any space location may (in elliptic problems) influence the flow anywhere else, forcing at a given instant will affect the flow only in the future – there is no backward influence. Unsteady flows are, therefore, parabolic-like in time. This means that no conditions can be imposed on the solution (except at the boundaries) at any time after the initiation of the calculation, which has a strong influence on the choice of solution strategy. To be faithful to the nature of time, essentially all solution methods advance in time in a step-by-step or "marching" manner. These methods are very similar to ones applied to initial value problems for ordinary differential equations (ODEs) so we shall give a brief review of such methods in the next section.

6.2 Methods for Initial Value Problems in ODEs

6.2.1 Two-Level Methods

For initial value problems, it is sufficient to consider the first order ordinary differential equation with an initial condition:

$$\frac{\mathrm{d}\phi(t)}{\mathrm{d}t} = f(t, \phi(t)) \ ; \quad \phi(t_0) = \phi^0 \ . \tag{6.1}$$

The basic problem is to find the solution ϕ a short time Δt after the initial point. The solution at $t_1 = t_0 + \Delta t$, ϕ^1, can be regarded as a new initial condition and the solution can be advanced to $t_2 = t_1 + \Delta t$, $t_3 = t_2 + \Delta t, \ldots$ etc.

The simplest methods can be constructed by integrating Eq. (6.1) from t_n to $t_{n+1} = t_n + \Delta t$:

$$\int_{t_n}^{t_{n+1}} \frac{d\phi}{dt} \, dt = \phi^{n+1} - \phi^n = \int_{t_n}^{t_{n+1}} f(t, \phi(t)) \, dt \, , \tag{6.2}$$

where we use the shorthand notation $\phi^{n+1} = \phi(t_{n+1})$. This equation is exact. However, the right hand side cannot be evaluated without knowing the solution so some approximation is necessary. The mean value theorem of calculus guarantees that if the integrand is evaluated at the proper point $t = \tau$ between t_n and t_{n+1}, the integral is equal to $f(\tau, \phi(\tau)) \, \Delta t$ but this is of little use since τ is unknown. We therefore use some approximate numerical quadrature to evaluate the integral.

Four relatively simple procedures are given below; a geometric picture is provided in Fig. 6.1.

If the integral on the right hand side of Eq. (6.2) is estimated using the value of the integrand at the initial point, we have:

$$\phi^{n+1} = \phi^n + f(t_n, \phi^n) \, \Delta t \, , \tag{6.3}$$

which is known as the *explicit* or *forward Euler* method.

If, instead, we use the final point in estimating the integral, we obtain the *implicit* or *backward Euler* method:

$$\phi^{n+1} = \phi^n + f(t_{n+1}, \phi^{n+1}) \, \Delta t \, . \tag{6.4}$$

Still another method can be obtained by using the midpoint of the interval:

$$\phi^{n+1} = \phi^n + f(t_{n+\frac{1}{2}}, \phi^{n+\frac{1}{2}}) \, \Delta t \, , \tag{6.5}$$

which is known as the *midpoint rule* and may be regarded as the basis of an important method for solving partial differential equations – the leapfrog method.
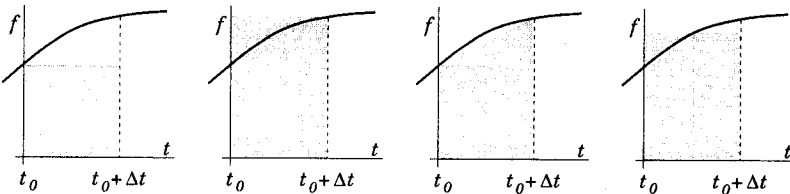
Fig. 6.1. Approximation of the time integral of $f(t)$ over an interval Δt (from left to right: explicit Euler, implicit Euler, trapezoidal rule and midpoint rule, respectively)

Finally, one can use straight line interpolation between the initial and final points to construct the approximation:

$$\phi^{n+1} = \phi^n + \frac{1}{2} \left[f(t_n, \phi^n)) + f(t_{n+1}, \phi^{n+1}) \right] \Delta t \, , \tag{6.6}$$

which is called the *trapezoid rule* and is the basis for a popular method of solving partial differential equations – the Crank-Nicolson method.

Collectively, these methods are called two-level methods because they involve the values of the unknown at only two times (the midpoint rule may or may not be regarded as a two-level method depending on what further approximations are employed). Analysis of these methods can be found in texts on the numerical solution of ordinary differential equations (see the bibliography) and will not be repeated here. We shall simply review some of their most important properties. First we note that all of the methods but the first require the value of $\phi(t)$ at some point other than $t = t_n$ (which is the initial point of the integration interval at which the solution is known). Therefore, for these methods, the right hand side cannot be calculated without further approximation or iteration. Thus, the first method belongs to the class called *explicit* methods while all of the others are *implicit*.

All methods produce good solutions if Δt is small. However, the behavior of methods for large step size is important because, in problems with widely varying time scales (including many problems in fluid mechanics), the goal is often to compute the slow, long term behavior of the solution and the short time scales are merely a nuisance. Problems with a wide range of time scales are called *stiff* and are the greatest difficulty one faces in the solution of ordinary differential equations. It is therefore important to inquire about the behavior of methods when the step size is large. This raises the issue of *stability*.

There are a number of definitions of stability in the literature. We shall use a rough definition that calls a method stable if it produces a bounded solution when the solution of the underlying differential equation is also bounded. For the explicit Euler method, stability requires:

$$\left|1 + \Delta t \frac{\partial f(t, \phi)}{\partial \phi}\right| < 1 \,, \tag{6.7}$$

which, if $f(t, \phi)$ is allowed to have complex values, requires that $\Delta t\, \partial f(t, \phi)/\partial \phi$ be restricted to the unit circle with center at -1. (Complex values must be considered because higher order systems may have complex eigenvalues. Only values with zero or negative real part are of interest because they lead to bounded solutions.) A method with this property is called *conditionally stable*; for real values of f, Eq. (6.7) reduces to (see Eq. (6.1)):

$$\left|\Delta t \frac{\partial f(t, \phi)}{\partial \phi}\right| < 2 \,. \tag{6.8}$$

All of the other methods defined above are *unconditionally stable* i.e. they produce bounded solutions for any time step if $\partial f(t, \phi)/\partial \phi < 0$. However, the implicit Euler method tends to produce smooth solutions even when Δt is very large while the trapezoid rule frequently yields solutions which oscillate with little damping. Consequently, the implicit Euler method tends to behave

well, even for non-linear equations, while the trapezoid rule may be unstable for non-linear problems.

Finally, the question of accuracy needs to be considered. On a global scale, it is difficult to say much about this issue due to the incredible variety of equations that one might need to deal with. For a single small step it is possible to use Taylor series to show that the explicit Euler method, starting with the known solution at t_n, yields the solution at time $t_n + \Delta t$ with an error proportional to $(\Delta t)^2$. However, as the number of steps required to compute to some finite final time $t = t_0 + T$ is inversely proportional to Δt, and an error is incurred on each step, the error will at the end be proportional to Δt itself. Therefore, explicit Euler method is a first order method. The implicit Euler method is also a first order method, while the trapezoid and midpoint rule methods have errors proportional to $(\Delta t)^2$ and are therefore second order methods. It can be shown that second order is the highest order achievable by a two-level scheme.

It is important to note that the order of a method is not the sole indicator of its accuracy. While it is true that, for small enough step size, a high order method will have a smaller error than a low order one, it is also true that two methods of the same order may have errors that differ by as much as an order of magnitude. The order determines only the *rate* at which the error goes to zero as the step size goes to zero, and this only after the step size has become small enough. 'Small enough' is both method and problem dependent and cannot be determined a priori.

When the step size is small enough, one can estimate the discretization error in the solution by comparing solutions obtained using two different step sizes. This method, known as *Richardson extrapolation*, has been described in Chap. 3, and applies to both spatial and temporal discretization errors. The error can also be estimated by analyzing the difference in solutions produced by two schemes of different order; this will be discussed in Chap. 11.

6.2.2 Predictor-Corrector and Multipoint Methods

The properties that we have found for two-level methods are quite general. Explicit methods are very easy to program and use little computer memory and computation time per step but are unstable if the time step is large. On the other hand, implicit methods require iterative solution to obtain the values at the new time step. This makes them harder to program and they use more computer memory and time per time step, but they are much more stable. (The implicit methods described above are unconditionally stable; this is not true of all implicit methods but they are generally more stable than their explicit counterparts.) One might ask whether it is possible to combine the best of the two methods. Predictor-corrector methods are an attempt to do this.

A wide variety of predictor-corrector methods has been developed; for the present, we shall give just one, which is so well-known that it is often called

the predictor-corrector method. In this method, the solution at the new time step is *predicted* using the explicit Euler method:

$$\phi^*_{n+1} = \phi^n + f(t_n, \phi^n)\, \Delta t \,, \tag{6.9}$$

where the $*$ indicates that this is not the final value of the solution at t_{n+1}. Rather, the solution is *corrected* by applying the trapezoid rule using ϕ^*_{n+1} to compute the derivative:

$$\phi^{n+1} = \phi^n + \frac{1}{2}\Big[f(t_n, \phi^n) + f(t_{n+1}, \phi^*_{n+1})\Big]\, \Delta t \,. \tag{6.10}$$

This method can be shown to be second order accurate (the accuracy of the trapezoid rule) but has roughly the stability of the explicit Euler method. One might think that by iterating the corrector, the stability might be improved but this turns out not to be the case because this iteration procedure converges to the trapezoid rule solution only if Δt is small enough.

This predictor-corrector method belongs to the two-level family, for which the highest accuracy possible is second order. For higher-order approximations one must use information at more points. The additional points may be ones at which data has already been computed or points between t_n and t_{n+1} which are used strictly for computational convenience; the former are called multipoint methods, the latter, Runge-Kutta methods. Here, we shall present multipoint methods; Runge-Kutta methods are presented in the next section.

The best known multipoint methods, the Adams methods, are derived by fitting a polynomial to the derivatives at a number of points in time. If a Lagrange polynomial is fit to $f(t_{n-m}, \phi^{n-m})$, $f(t_{n-m+1}, \phi^{n-m+1})$, $\ldots$, $f(t_n, \phi^n)$, and the result is used to compute the integral in Eq. (6.2), we obtain an explicit method of order $m + 1$; methods of this type are called *Adams-Bashforth* methods. For the solution of partial differential equations, only the lower order methods are used. The first order method is explicit Euler while the second and third order methods are:

$$\phi^{n+1} = \phi^n + \frac{\Delta t}{2}\Big[3\, f(t_n, \phi^n) - f(t_{n-1}, \phi^{n-1})\Big] \tag{6.11}$$

and

$$\phi^{n+1} = \phi^n + \frac{\Delta t}{12}\Big[23\, f(t_n, \phi^n) - 16\, f(t_{n-1}, \phi^{n-1}) + 5\, f(t_{n-2}, \phi^{n-2})\Big] \,. \tag{6.12}$$

If data at t_{n+1} is included in the interpolation polynomial, implicit methods, known as *Adams-Moulton* methods, are obtained. The first order method is implicit Euler, the second order one is trapezoid rule and the third order method is:

$$\phi^{n+1} = \phi^n + \frac{\Delta t}{12}\Big[5\, f(t_{n+1}, \phi^{n+1}) + 8\, f(t_n, \phi^n) - f(t_{n-1}, \phi^{n-1})\Big] \,. \tag{6.13}$$

A common method uses the $(m-1)$st order Adams-Bashforth method as a predictor and the mth order Adams-Moulton method as a corrector. Thus, predictor-corrector methods of any order can be obtained.

The multipoint approach has the advantage that it is relatively easy to construct methods of any order. These methods are also easy to use and program. A final advantage is that they require only one evaluation of the derivative per time step (which may be very complicated, especially in applications involving partial differential equations), making them relatively cheap. (The values of $f(t, \phi(t))$ are required several times, but they can be stored once calculated, so that only one evaluation per time step is required.) Their principal disadvantage is that, because they require data from many points prior to the current one, they cannot be started using only data at the initial time point. One has to use other methods to get calculation started. One approach is to use a small step size and a lower order method (so that the desired accuracy is achieved) and slowly increase the order as more data become available.

These methods are the basis for many accurate ordinary differential equation solvers. In these solvers, error estimators are used to determine the accuracy of the solution at every step. If the solution is not accurate enough, the order of the method is increased up to the maximum order allowed by the program. On the other hand, if the solution is much more accurate than necessary, the order of the method might be reduced to save computing time. Because the step size is difficult to change in multipoint methods, this is done only when the maximum order of the method has already been reached.[1]

Because multipoint methods use data from several time steps, they may produce non-physical solutions. Space does not permit inclusion of the analysis here but it is worth noting that the instabilities of multipoint methods are most often due to the non-physical solutions. These may be suppressed, but not entirely, by a careful choice of the starting method. It is common for these methods to give an accurate solution for some time and then begin to behave badly as the non-physical component of the solution grows. A common remedy for this problem is to restart the method every so often, a trick that is effective but may reduce the accuracy and/or the efficiency of the scheme.

6.2.3 Runge-Kutta Methods

The difficulties in starting multipoint methods can be overcome by using points between t_n and t_{n+1} rather than earlier points. Methods of this kind are called Runge-Kutta methods. We shall give just two examples.

---

[1] The quadrature approximations used above assume a uniform time step; if the time step is allowed to vary, the coefficients multiplying the function values at different time levels become complicated functions of step sizes, as we saw in Chap. 3 for finite differences in space.

The second order Runge-Kutta method consists of two steps. The first may be regarded as a half-step predictor based on the explicit Euler method; it is followed by a midpoint rule corrector which makes the method second order:

$$\phi^*_{n+\frac{1}{2}} = \phi^n + \frac{\Delta t}{2} f(t_n, \phi^n) , \tag{6.14}$$

$$\phi^{n+1} = \phi^n + \Delta t\, f(t_{n+\frac{1}{2}}, \phi^*_{n+\frac{1}{2}}) . \tag{6.15}$$

This method is easy to use and is self-starting i.e. it requires no data other than the initial condition required by the differential equation itself. In fact, it is very similar in many ways to the predictor-corrector method described above.

Runge-Kutta methods of higher order have been developed; the most popular one is of fourth order. The first two steps of this method use an explicit Euler predictor and an implicit Euler corrector at $t_{n+\frac{1}{2}}$. This is followed by a midpoint rule predictor for the full step and a Simpson's rule final corrector that gives the method its fourth order. The method is:

$$\phi^*_{n+\frac{1}{2}} = \phi^n + \frac{\Delta t}{2} f(t_n, \phi^n) , \tag{6.16}$$

$$\phi^{**}_{n+\frac{1}{2}} = \phi^n + \frac{\Delta t}{2} f(t_{n+\frac{1}{2}}, \phi^*_{n+\frac{1}{2}}) , \tag{6.17}$$

$$\phi^*_{n+1} = \phi^n + \Delta t\, f(t_{n+\frac{1}{2}}, \phi^{**}_{n+\frac{1}{2}}) , \tag{6.18}$$

$$\begin{aligned}\phi^{n+1} = \phi^n + \frac{\Delta t}{6} \Big[& f(t_n, \phi^n) + 2 f(t_{n+\frac{1}{2}}, \phi^*_{n+\frac{1}{2}}) + \\ & 2 f(t_{n+\frac{1}{2}}, \phi^{**}_{n+\frac{1}{2}}) + f(t_{n+1}, \phi^*_{n+1}) \Big] .\end{aligned} \tag{6.19}$$

A number of variations on this method have been developed. In particular, there are several methods which add a fifth step of either fourth or fifth order to allow estimation of the error and, thereby, the possibility of automatic error control.

The major problem with Runge-Kutta methods is that it is somewhat difficult to develop methods of very high order and, as is readily seen from the methods given above, an nth order Runge-Kutta method requires that the derivative be evaluated n times per time step, making these methods more expensive than multipoint methods of comparable order. In partial compensation, the Runge-Kutta methods of a given order are more accurate (i.e. the coefficient of the error term is smaller) and more stable than the multipoint methods of the same order.

6.2.4 Other Methods

One can approximate the integrals on both sides of Eq. (6.2) by mean values over the integration interval.

An implicit three-level second order scheme can be constructed by integrating over a time interval Δt centered around t_{n+1} (i.e. from $t_{n+1} - \Delta t/2$ to $t_{n+1} + \Delta t/2$) and applying the midpoint rule to both the left and right hand sides of the equation. The time derivative at t_{n+1} can be approximated by differentiating a parabola forced through solutions at three time levels, t_{n-1}, t_n, and t_{n+1}:

$$\left(\frac{\mathrm{d}\phi}{\mathrm{d}t}\right)_{n+1} \approx \frac{3\,\phi^{n+1} - 4\,\phi^n + \phi^{n-1}}{2\,\Delta t} \,. \tag{6.20}$$

This leads to the following method:

$$\phi^{n+1} = \frac{4}{3}\phi^n - \frac{1}{3}\phi^{n-1} + \frac{2}{3}\,f(t_{n+1}, \phi^{n+1})\,\Delta t \,. \tag{6.21}$$

The scheme is implicit, since f is evaluated at the new time level. It is of second order and very easy to implement, but as it is implicit, it requires iteration at each time step.

6.3 Application to the Generic Transport Equation

We next consider application of some of the methods given above to the generic transport equation (1.28). In Chaps. 3 and 4, discretization of the convective and diffusive fluxes and source terms for steady problems was discussed. These terms can be treated in the same way for unsteady flows; however, the question of the time at which the fluxes and sources are to be evaluated must be answered.

If the conservation equation is rewritten in a form which resembles the ordinary differential equation (6.1), e.g.:

$$\frac{\partial(\rho\phi)}{\partial t} = -\mathrm{div}\,(\rho\phi\boldsymbol{v}) + \mathrm{div}\,(\Gamma\,\mathrm{grad}\,\phi) + q_\phi = f(t, \phi(t)) \,, \tag{6.22}$$

any method of time integration can be used. The function $f(t, \phi)$ represents the sum of convective, diffusive and source terms, all of which now appear on the right hand side of the equation. Since these terms are not known, we must use one of the quadrature approximations introduced above. The convective, diffusive and source terms are discretized using one of the methods presented in Chaps. 3 and 4 at one or more time levels. If an explicit method is used for time integration, these terms have to be evaluated only at times for which the solution is already known, so they can be calculated. For an implicit method,

the discretized right hand side of the above equation is required at the new time level, for which the solution is not known yet. Therefore, an algebraic system of equations, which differs from the one obtained for steady problems, must be solved. We shall analyze properties of some of the common schemes when applied to a 1D problem below; solutions of a 1D and a 2D problem will be discussed in the examples section.

6.3.1 Explicit Methods

Explicit Euler Method. The simplest method is explicit Euler in which all fluxes and sources are evaluated using known values at t_n. In the equation for a CV or grid point, the only unknown at the new time level is the value at that node; the neighbor values are all evaluated at earlier time levels. Thus one can explicitly calculate the new value of the unknown at each node.

In order to study properties of the explicit Euler and other simple schemes, we consider the 1D version of Eq. (6.22) with constant velocity, constant fluid properties, and no source terms:

$$\frac{\partial \phi}{\partial t} = -u \frac{\partial \phi}{\partial x} + \frac{\Gamma}{\rho} \frac{\partial^2 \phi}{\partial x^2} . \tag{6.23}$$

This equation is often used in the literature as a model equation for the Navier-Stokes equations. It is the time dependent version of the equation (3.61) used to illustrate methods for steady problems. Like that equation, it assumes that the important balance is between advection and streamwise diffusion, a balance that rarely occurs in real flows. For this reason, one must be careful about extending what is learned from this equation to the Navier-Stokes equations. Despite this important shortcoming, we can learn something by considering Eq. (6.23).

We first assume that the spatial derivatives are approximated using CDS and that the grid is uniform in x-direction. In this case the same algebraic equation results from both FD and FV discretizations. The new variable value, ϕ_i^{n+1}, is:

$$\phi_i^{n+1} = \phi_i^n + \left[-u \frac{\phi_{i+1}^n - \phi_{i-1}^n}{2\,\Delta x} + \frac{\Gamma}{\rho} \frac{\phi_{i+1}^n + \phi_{i-1}^n - 2\,\phi_i^n}{(\Delta x)^2} \right] \Delta t \, , \tag{6.24}$$

which can be rewritten:

$$\phi_i^{n+1} = (1 - 2d)\,\phi_i^n + \left(d - \frac{c}{2}\right) \phi_{i+1}^n + \left(d + \frac{c}{2}\right) \phi_{i-1}^n \, , \tag{6.25}$$

where we introduced the dimensionless parameters:

$$d = \frac{\Gamma\,\Delta t}{\rho (\Delta x)^2} \quad \text{and} \quad c = \frac{u\,\Delta t}{\Delta x} . \tag{6.26}$$

The parameter d is the ratio of time step Δt to the characteristic diffusion time $\rho(\Delta x)^2/\Gamma$, which is roughly the time required for a disturbance to be transmitted by diffusion over a distance Δx. The second quantity (c) is the ratio of time step Δt to the characteristic convection time, $u/\Delta x$, the time required for a disturbance to be convected a distance Δx. This ratio is called *Courant number* and is one of the key parameters in computational fluid dynamics.

If ϕ were the temperature (a possibility), Eq. (6.25) would need to satisfy several conditions. By virtue of diffusion, an increase in temperature at any of the three points x_{i-1}, x_i and x_{i+1} at the old time level should increase the temperature at point x_i at the new time level. The same can be said for the points x_{i-1} and x_i with respect to convection, assuming $u > 0$. If ϕ represents the concentration of a substance, it should not be negative.

The possibility that some of the coefficients of ϕ_i^{n-1} and ϕ_{i+1}^{n-1} in Eq. (6.25) can become negative should alert us to possible trouble and demands a more detailed analysis. To the extent possible, we want this analysis to mimic the one used for ordinary differential equations. A simple way to do this was invented by *von Neumann* for whom the method is named. He argued that the boundary conditions are rarely the cause of problems (there are exceptions that are not relevant here) so why not ignore them altogether? If that is done, the analysis is simplified. Since this method of analysis can be applied to essentially all of the methods discussed in this chapter, we shall describe it in a little detail; for further details, see the book by Strikwerda (1983).

In essence the idea can be arrived at as follows. The set of Eqs. (6.25) can be written in matrix form:

$$\phi^{n+1} = A\phi^n \; , \tag{6.27}$$

where the elements of the tridiagonal matrix A can be derived by inspection of Eq. (6.25). This equation gives the solution at the new step in terms of the solution at the previous step. The solution at t_{n+1} can thus be obtained by repetitive multiplication of the initial solution ϕ^0 by the matrix A. The question is: do the differences between solutions at successive time steps (for non-varying boundary conditions), measured in any convenient way, increase, decrease, or stay the same as n is increased? For example, one measure is the norm:

$$\epsilon = ||\phi^n - \phi^{n-1}|| = \sqrt{\sum_i (\phi_i^n - \phi_i^{n-1})^2} \; . \tag{6.28}$$

The differential equation requires this quantity to decrease with time through the action of dissipation. Eventually, a steady-state solution will be obtained if the boundary conditions do not vary. Naturally, we would like the numerical method to preserve this property of the exact equations.

This issue is closely connected with the eigenvalues of the matrix A. If some of them are greater than 1, it is not difficult to show that ϵ will grow

with n while, if all of them are smaller than 1, ϵ will decay. Normally, the eigenvalues of a matrix are difficult to estimate and, for a problem more difficult than this, we would be in serious trouble. The saving feature of this problem is that, because each diagonal of the matrix is constant, the eigenvectors are easily found. They can be represented in terms of sines and cosines but it is simpler to use the complex exponential form:

$$\phi_j^n = \sigma^n \mathrm{e}^{\mathrm{i}\alpha j} , \tag{6.29}$$

where $\mathrm{i} = \sqrt{-1}$ and α is a wavenumber that can be chosen arbitrarily; the choice will be discussed below. If Eq. (6.29) is substituted into Eq. (6.25), the complex exponential term $\mathrm{e}^{\mathrm{i}\alpha j}$ is common to every term and can be removed and we obtain an explicit expression for the eigenvalue σ:

$$\sigma = 1 + 2d(\cos\alpha - 1) + \mathrm{i}\,2c\sin\alpha . \tag{6.30}$$

The magnitude of this quantity is what is important. Since the magnitude of a complex quantity is the sum of the squares of the real and imaginary parts, we have:

$$\sigma^2 = [1 + 2d(\cos\alpha - 1)]^2 + 4c^2\sin^2\alpha . \tag{6.31}$$

We now investigate the conditions for σ^2 to be smaller than unity.

Since there are two independent parameters in the expression for σ, it is simplest to consider special cases first. When there is no diffusion ($d = 0$), $\sigma > 1$ for any α and this method is unstable for any value of c, i.e. the method is *unconditionally unstable*, rendering it useless. On the other hand, when there is no convection ($c = 0$), we find that σ is maximum when $\cos\alpha = -1$ so the method is stable provided $d < \frac{1}{2}$ i.e. it is conditionally stable.

The requirement that the coefficients of all old nodal values be positive leads to similar conclusions: $d < 0.5$ and $c < 2d$. The first condition leads to the limit on Δt:

$$\Delta t < \frac{\rho(\Delta x)^2}{2\Gamma} . \tag{6.32}$$

The second requirement imposes no limit on the time step, but gives a relation between convection and diffusion coefficients:

$$\frac{\rho u\,\Delta x}{\Gamma} < 2 \quad \text{or} \quad \mathrm{Pe}_{\mathrm{cell}} < 2 , \tag{6.33}$$

i.e., the cell Peclet number should be smaller than two. This has already been mentioned as a sufficient (but not necessary) condition for boundedness of solutions obtained using CDS for convective fluxes.

Since the method is based on a combination of the explicit Euler method for ordinary differential equations and the central difference approximation for the spatial derivatives, it inherits the accuracy of each. The method is therefore first order in time and second order in space. The requirement that

$d < 0.5$ means that, each time the spatial mesh is halved, the time step has to be reduced by a factor of four. This makes the scheme unsuitable for problems which do not require high temporal resolution (slowly varying solutions, solutions approaching steady state); these are normally the cases in which one would like to use first order methods in time. An example which illustrates the stability problem will be shown below.

The problem of instability in connection with the condition of Eq. (6.33) was recognized in the 1920's by Courant and Friedrichs and they suggested a cure which is still used today. They noted that, for problems dominated by convection, it is possible for the coefficient of ϕ_{i+1}^{n-1} in Eq. (6.25) to be negative and suggested that the problem could be cured by using *upwind differences.* Instead of evaluating the convective term by a CDS approximation as we have done above, we use UDS (see Chap. 3). We then get, in place of Eq. (6.24):

$$\phi_i^{n+1} = \phi_i^n + \left[-u \frac{\phi_i^n - \phi_{i-1}^n}{\Delta x} + \frac{\Gamma}{\rho} \frac{\phi_{i+1}^n + \phi_{i-1}^n - 2\,\phi_i^n}{(\Delta x)^2} \right] \Delta t\,. \tag{6.34}$$

which yields, in place of Eq. (6.25):

$$\phi_i^{n+1} = (1 - 2d - c)\phi_i^n + d\phi_{i+1}^n + (d + c)\phi_{i-1}^n\,. \tag{6.35}$$

Since the coefficients of the neighbor values are always positive, they cannot contribute to unphysical behavior of instability. However, the coefficient ϕ_i^n can be negative, thus creating a potential problem. For this coefficient to be positive, the time step should satisfy the following condition:

$$\Delta t < \frac{1}{\dfrac{2\Gamma}{\rho(\Delta x)^2} + \dfrac{u}{\Delta x}}\,. \tag{6.36}$$

When convection is negligible, the restriction on the time step required for stability is the same as Eq. (6.32). For negligible diffusion, the criterion to be satisfied is:

$$c < 1 \quad \text{or} \quad \Delta t < \frac{\Delta x}{u} \tag{6.37}$$

i.e. the Courant number should be smaller than unity.

One can also apply the von Neumann stability analysis to this problem; the analysis is in accord with the conclusions just reached. Thus, unlike the central difference method, the upwind approximation provides some stability which, combined with its ease of use, made this type of method very popular for many years. It continues to be used today. When both convection and diffusion are present, the stability criterion is more complicated. Rather than dealing with this complexity, most people require that each individual criterion be satisfied, a condition that may be a little more restrictive than necessary but is safe.

This method has first order truncation errors in both space and time and requires very small step sizes in both variables if errors are to be kept small.

The restriction on the Courant number also has the interpretation that a fluid particle cannot move more than one grid length in a single time step. This restriction on the rate of information propagation appears very reasonable but can limit the rate of convergence when methods of this kind are employed for the solution of steady state problems.

Other explicit schemes may be based on other methods for ordinary differential equations. Indeed, all of the methods described earlier have been used at one time or another in CFD. A central difference based three time level method, known as the *leapfrog method*, will be described next.

Leapfrog Method. A commonly used three-level scheme is the leapfrog method; it is essentially the application of the midpoint rule integration to a time interval of size $2\Delta t$:

$$\phi_i^{n+1} = \phi_i^{n-1} + f(t_n, \phi^n)\, 2\Delta t \,. \tag{6.38}$$

When it is applied to the generic transport equation, Eq. (6.23), in which CDS is used for spatial discretization, we obtain:

$$\phi_i^{n+1} = \phi_i^{n-1} + \left[-u \frac{\phi_{i+1}^n - \phi_{i-1}^n}{2\,\Delta x} + \frac{\Gamma}{\rho} \frac{\phi_{i+1}^n + \phi_{i-1}^n - 2\,\phi_i^n}{(\Delta x)^2} \right] 2\Delta t \,. \tag{6.39}$$

In terms of the dimensionless coefficients d and c, the above equation reads:

$$\phi_i^{n+1} = \phi_i^{n-1} - 4d\phi_i^n + (2d - c)\phi_{i+1}^n + (2d + c)\phi_{i-1}^n \,. \tag{6.40}$$

In this method, the heuristic requirements for a physically realistic simulation of heat conduction are never satisfied, since the coefficient of ϕ_i^n is unconditionally negative! Indeed, stability analysis shows this scheme to be unconditionally unstable and it appears not to be useful for the numerical solution of unsteady problems. However, the instability is very weak if the time step is small and wave-like solutions are very weakly damped compared to other methods. For these reasons, this method is actually used (with some tricks to stabilize it) in a number of applications, especially in meteorology and oceanography.

One way to stabilize the scheme is to use the approximation:

$$\phi_i^n \approx \frac{1}{2} \left(\phi_i^{n-1} + \phi_i^{n+1} \right) \,, \tag{6.41}$$

which is a central difference approximation in time. This approximation is known as the *DuFort-Frankel* method and recasts the above equation into the following form:

$$(1 + 2d)\phi_i^{n+1} = (1 - 2d)\phi_i^{n-1} + (2d - c)\phi_{i+1}^n + (2d + c)\phi_{i-1}^n \,. \tag{6.42}$$

Surprisingly, the scheme is now unconditionally stable, but the above approximation introduces another truncation error, which has the unusual property

of being proportional to $(\Delta t/\Delta x)^2$. This term stems from the substitution (6.41) and is an undesirable feature of the method because the method is consistent (that is, it yields a solution of the partial differential equation in the limit of small step sizes) only if Δt tends faster to zero than Δx.

To obtain higher order accuracy in time, with reasonable limits on stability, a number of authors have used the Adams-Bashforth second and third order methods and, more commonly, third and fourth order Runge-Kutta methods.

6.3.2 Implicit Methods

Implicit Euler Method. If stability is a prime requirement, the analysis of methods for ordinary differential equations suggests use of the backward or implicit Euler method. Applied to the generic transport equation (6.23), with the CDS approximation to the spatial derivatives, it gives:

$$\phi_i^{n+1} = \phi_i^n + \left[-u \frac{\phi_{i+1}^{n+1} - \phi_{i-1}^{n+1}}{2\,\Delta x} + \frac{\Gamma}{\rho} \frac{\phi_{i+1}^{n+1} + \phi_{i-1}^{n+1} - 2\,\phi_i^{n+1}}{(\Delta x)^2} \right] \Delta t \,, \tag{6.43}$$

or, rearranged:

$$(1+2d)\,\phi_i^{n+1} + \left(\frac{c}{2} - d\right)\phi_{i+1}^{n+1} + \left(-\frac{c}{2} - d\right)\phi_{i-1}^{n+1} = \phi_i^n \,. \tag{6.44}$$

In this method, all of the fluxes and source terms are evaluated in terms of the unknown variable values at the new time level. The result is a system of algebraic equations very similar to the one obtained for steady problems; actually, the only difference lies in an additional contribution to the coefficient A_{P} and to the source term Q_{P}, which stem from the unsteady term. The above equations may be written:

$$A_{\mathrm{P}}\phi_i^{n+1} + A_{\mathrm{E}}\phi_{i+1}^{n+1} + A_{\mathrm{W}}\phi_{i-1}^{n+1} = Q_{\mathrm{P}} \,, \tag{6.45}$$

where

$$\begin{aligned} A_{\mathrm{E}} &= \frac{\rho u}{2\,\Delta x} - \frac{\Gamma}{(\Delta x)^2} \,; \quad A_{\mathrm{W}} = -\frac{\rho u}{2\,\Delta x} - \frac{\Gamma}{(\Delta x)^2} \,; \\ A_{\mathrm{P}} &= -(A_{\mathrm{E}} + A_{\mathrm{W}}) + \frac{\rho}{\Delta t} \,; \quad Q_{\mathrm{P}} = \frac{\rho}{\Delta t}\,\phi_i^n \end{aligned} \tag{6.46}$$

As was the case for ordinary differential equations, use of the implicit Euler method allows arbitrarily large time steps to be taken; this property is useful in studying flows with slow transients or steady flows. Problems may arise when CDS is used on coarse grids (if the Peclet number is too large in regions of strong change in variable gradient); oscillatory solutions are produced but the scheme remains stable.

The shortcomings of this method are its first order truncation error in time and the need to solve a large coupled set of equations at each time step.

It also requires much more storage than the explicit scheme, since the entire coefficient matrix A and the source vector have to be stored. The advantage is the possibility of using a large time step, which may result in a more efficient procedure despite the shortcomings.

This method is especially useful for solving steady flow problems. As noted in the previous chapter, the solution of coupled non-linear equations may require use of under-relaxation and nested (inner and outer) iterations, see Sect. 5.4.2. There is a strong similarity between the algebraic equations resulting from the use of under-relaxation when solving steady problems and those resulting from implicit Euler scheme applied to unsteady equations. Both under-relaxation and implicit time discretization result in an additional source term and a contribution to the central coefficient $A_{\rm P}$. The following relation between the under-relaxation factor α_ϕ and time step Δt can be derived by requiring that the contributions be same in both cases (see Eqs. (5.70) and (6.46)):

$$\Delta t = \frac{\rho \alpha_\phi \, \Delta \Omega}{A_{\rm P}(1 - \alpha_\phi)} \quad \text{or} \quad \alpha_\phi = \frac{A_{\rm P} \Delta t}{A_{\rm P} \Delta t + \rho \, \Delta \Omega} \, . \tag{6.47}$$

In the iteration at the new time step, the best initial guess is the converged solution at the preceding step. If the final steady state is the only result of interest and the details of the development from the initial guess to the final stage are not of importance, it might suffice to perform only one iteration per time step. Then one does not have to store the old solution – it is needed only to assemble the matrix and source terms. The major difference between using pseudo-time marching and under-relaxation is that using the same time step for all CVs is equivalent to using a variable under-relaxation factor; conversely, use of a constant under-relaxation factor is equivalent to applying a different time step to each control volume.

It is important to note that, if only one iteration is performed at each time step, the scheme may not retain all of the stability of the implicit Euler method. There may then be a limitation on the time step that can be employed (when under-relaxation is used in outer iterations, the choice of the parameter α_ϕ is also limited and certainly has to be smaller than unity).

Crank-Nicolson Method. The second order accuracy of the trapezoid rule method and its relative simplicity suggest its application to partial differential equations when time accuracy is of importance. It is then known as the Crank-Nicolson method. In particular, when applied to the 1D generic transport equation with CDS discretization of spatial derivatives one has:

$$\phi_i^{n+1} = \phi_i^n + \frac{\Delta t}{2} \left[-u \, \frac{\phi_{i+1}^{n+1} - \phi_{i-1}^{n+1}}{2 \Delta x} + \frac{\Gamma}{\rho} \, \frac{\phi_{i+1}^{n+1} + \phi_{i-1}^{n+1} - 2\phi_i^{n+1}}{(\Delta x)^2} \right] +$$

$$\frac{\Delta t}{2} \left[-u \, \frac{\phi_{i+1}^{n} - \phi_{i-1}^{n}}{2 \Delta x} + \frac{\Gamma}{\rho} \, \frac{\phi_{i+1}^{n} + \phi_{i-1}^{n} - 2\phi_i^{n}}{(\Delta x)^2} \right] \, . \tag{6.48}$$

The scheme is implicit; the contributions from fluxes and sources at the new time level give rise to a coupled set of equations similar to those of implicit Euler scheme. The above equation can be written as:

$$A_P \phi_i^{n+1} + A_E \phi_{i+1}^{n+1} + A_W \phi_{i-1}^{n+1} = Q_i^t \,, \tag{6.49}$$

where:

$$\begin{aligned} A_E &= \frac{\rho u}{4\,\Delta x} - \frac{\Gamma}{2(\Delta x)^2} \,; \quad A_W = -\frac{\rho u}{4\,\Delta x} - \frac{\Gamma}{2(\Delta x)^2} \,, \\ A_P &= \frac{\rho}{\Delta t} - (A_E + A_W) \,, \\ Q_i^t &= \left(A_W + A_E + \frac{\rho}{\Delta t}\right) \phi_i^n - A_E \phi_{i+1}^n - A_W \phi_{i-1}^n \,. \end{aligned} \tag{6.50}$$

The term Q_i^t represents an "additional" source term, which contains the contribution from the previous time level; it remains constant during iterations at the new time level. The equation may also contain a source term dependent on the new solution, so the above term needs to be stored separately.

This scheme requires very little more computational effort per step than the first order implicit Euler scheme. Von Neumann stability analysis shows that the scheme is unconditionally stable, but oscillatory solutions (and even instability) are possible for large time steps. This may be attributed to the possibility of the coefficient of ϕ_i^{n-1} becoming negative at large Δt, but is guaranteed to be positive if $\Delta t < \rho(\Delta x)^2/\Gamma$, which is twice the maximum step size allowed by the explicit Euler method. In practice, much larger time steps can be used without producing oscillations; the limit is problem dependent.

This scheme can be regarded as an equal blend of first order explicit and implicit Euler schemes. Only for equal blending is second order accuracy obtained; for other blending factors, which may vary in space and time, the method remains first order accurate. The stability is increased if the implicit contribution is increased, but the accuracy is reduced.

Three Time Level Method. A fully implicit scheme of second order accuracy can be obtained by using a quadratic backward approximation in time, as described in Sect. 6.2.4. For the 1D generic transport equation and CDS discretization in space we obtain:

$$\begin{aligned} &\rho \frac{3\,\phi_i^{n+1} - 4\,\phi_i^n + \phi_i^{n-1}}{2\Delta t} \Delta t = \\ &\left[-\rho u \frac{\phi_{i+1}^{n+1} - \phi_{i-1}^{n+1}}{2\Delta x} + \Gamma \frac{\phi_{i+1}^{n+1} + \phi_{i-1}^{n+1} - 2\phi_i^{n+1}}{(\Delta x)^2} \right] \Delta t \,. \end{aligned} \tag{6.51}$$

The resulting algebraic equation can be written:

$$A_P \phi_i^{n+1} + A_E \phi_{i+1}^{n+1} + A_W \phi_{i-1}^{n+1} = \frac{2\rho}{\Delta t} \phi_i^n - \frac{\rho}{2\,\Delta t} \phi_i^{n-1} \,. \tag{6.52}$$

The coefficients $A_{\rm E}$ and $A_{\rm W}$ are the same as in case of implicit Euler scheme, see Eq. 6.46. The central coefficient has now a stronger contribution from the time derivative:

$$A_{\rm P} = -(A_{\rm E} + A_{\rm W}) + \frac{3\rho}{2\,\Delta t} , \tag{6.53}$$

and the source term contains contribution from time t_{n-1}, see Eq. 6.52.

This scheme is easier to implement than the Crank-Nicolson scheme; it is also less prone to producing oscillatory solutions, although this may happen with large values of Δt. One has to store the variable values from three time levels, but the memory requirements are the same as for the Crank-Nicolson scheme. The scheme is second order accurate in time, but for small time steps, this method is less accurate than the Crank-Nicolson method (the truncation error is four times as large as in the Crank-Nicolson method). One can show that this scheme is unconditionally stable. We also see from Eq. (6.52) that the coefficient of the old value at node i is always positive; however, the coefficient of the value at t_{n-1} is always negative, which is why the scheme may produce oscillatory solutions if the steps are large.

This scheme can be blended with the first order implicit Euler scheme. Only the contributions to the central coefficient and source term need be modified in the manner of the deferred correction approach described in Sect. 5.6. This is useful when starting the calculation, since only one old level solution is available. Also, if one is after a steady state solution, switching to implicit Euler scheme ensures stability and allows large time steps to be used. Blending in a small amount of the first order scheme helps prevent oscillations which contributes to the esthetics of the solution (the accuracy is no better without oscillations, but it looks nicer graphically). If oscillations do occur, one has to reduce the time step as the oscillations are an indication of large temporal discretization errors. This comment does not apply to schemes which are only conditionally stable.

6.3.3 Other Methods

The schemes described above are the ones most often used in general purpose CFD codes. For special purposes, for example, in large eddy and direct simulation of turbulence, one often uses higher-order schemes, such as third or fourth order Runge-Kutta or Adams methods. Usually, higher-order temporal discretization is used when the spatial discretization is also of higher order, which is the case when the solution domain is of regular shape so that higher-order methods in space are easy to apply. Application of higher-order methods for ordinary differential equations to CFD problems is straightforward.

6.4 Examples

In order to demonstrate the performance of some of the methods described above, we look first at the unsteady version of the example problem of Chap. 3. The problem to be solved is given by Eq. (6.23), with the following initial and boundary conditions: at $t = 0$, $\phi_0 = 0$; for all subsequent times, $\phi = 0$ at $x = 0$, $\phi = 1$ at $x = L = 1$, $\rho = 1$, $u = 1$ and $\Gamma = 0.1$. Since the boundary conditions do not change in time, the solution develops from the initial zero field to the steady state solution given in Sect. 3.11. For spatial discretization, second order CDS is used. For temporal discretization, we use both explicit and implicit first order Euler methods, the Crank-Nicolson method and the fully implicit method with three time levels.

We first demonstrate what happens when the explicit Euler scheme (which is only conditionally stable) is used with time steps which violate the stability condition. Figure 6.2 shows the evolution of the solution over a small time period calculated using a time step slightly below and another slightly above the critical value given by Eq. (6.32). When the time step is larger than the critical value, oscillations are generated which grow unboundedly with time. A few time steps later than the last solution shown in Fig. 6.2, the numbers become too large to be handled by the computer. With implicit schemes, no problems occured even when very large time steps were used.
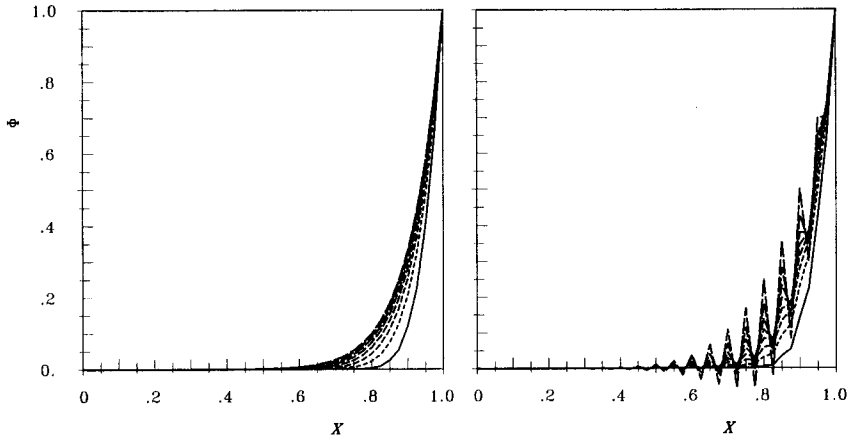
Fig. 6.2. Time evolution of the solution by explicit Euler method using time steps $\Delta t = 0.00325$ (left; $d < 0.5$ and $\Delta t = 0.003$ (right, $d > 0.5$)

In order to investigate the accuracy of temporal discretization, we look at the solution at the node at $x = 0.95$ of a uniform grid with 41 nodes ($\Delta x = 0.025$) at time $t = 0.01$. We performed calculations up to that time using 5, 10, 20 and 40 time steps with the four above mentioned schemes.

The convergence of ϕ at $x = 0.95$ for $t = 0.01$ as the time step size is reduced is shown in Fig. 6.3. The implicit Euler and the three level method under-predict, while the explicit Euler and the Crank-Nicolson method over-predict the correct value. All schemes show monotonic convergence towards the time step independent solution.
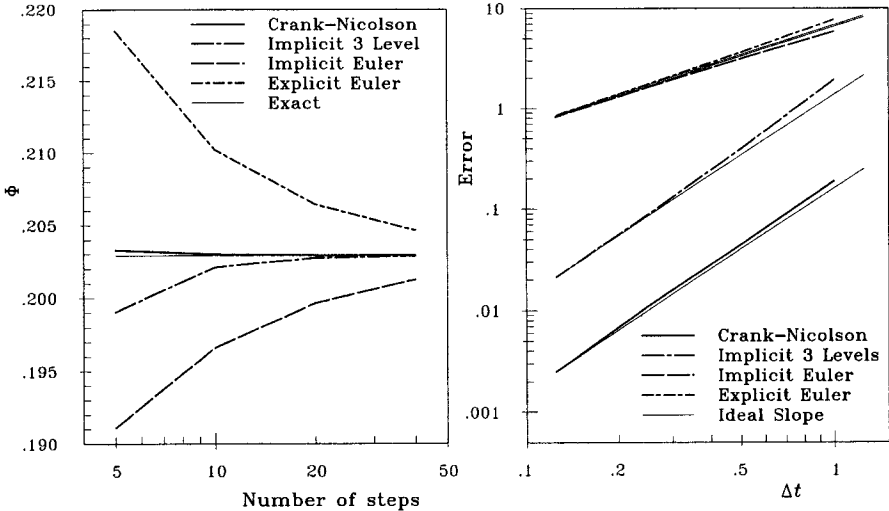
Fig. 6.3. Convergence of ϕ at $x = 0.95$ and $t = 0.01$ as the time step size is reduced (left) and temporal discretization errors (right) for various time integration schemes

Since we do not have an exact solution to compare with, we obtained an accurate reference solution at time $t = 0.01$ using the Crank-Nicolson scheme (the most accurate method) with $\Delta t = 0.0001$ (100 time steps). This solution is much more accurate than any of the above solutions so it can be treated as an exact solution for error estimation purposes. By subtracting solutions mentioned above from this reference solution, we obtained estimates of the temporal discretization error for each scheme and time step size. The spatial discretization error is the same in all cases and plays no role here. The errors thus obtained are plotted against time step size in Fig. 6.3.

The two Euler methods show the expected first order behavior: the error is reduced by one order of magnitude when the time step is reduced by the same amount. The two second order schemes show also the expected error reduction rate, which closely follow the ideal slope. However, the Crank-Nicolson method gives a more accurate solution since its initial error is much smaller. The three level scheme is started by the implicit Euler method, which resulted in a large initial error. Since, in this problem, the temporal variation is monotonic from the initial towards steady state, the initial error remains important throughout the solution. The error reduction rate is the same in

both methods, but the error level is in this case determined by its initial value.
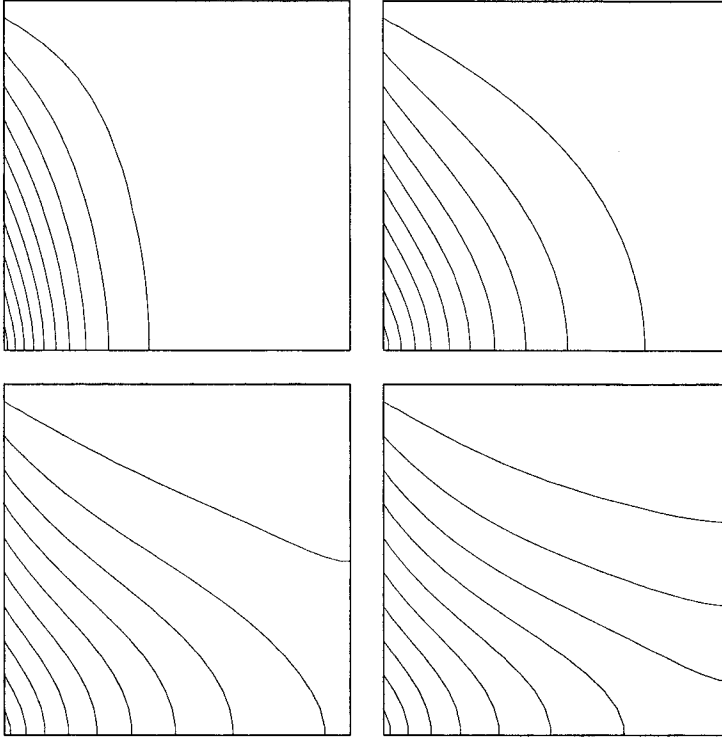
Fig. 6.4. Isotherms in the unsteady 2D problem at times $t = 0.2$ (upper left), $t = 0.5$ (upper right), $t = 1.0$ (lower left) and $t = 2.0$ (lower right), calculated on a uniform 20×20 CV grid using the CDS for spatial and the Crank-Nicolson method for temporal discretization

We next examine the 2D test case of Chap. 4, which involves heat transfer from a wall with prescribed temperature, exposed to a stagnation point flow, see Sect. 4.7. The initial solution is again $\phi_0 = 0$; the boundary conditions do not change with time and are the same as in the steady state problem investigated in Sect. 4.7, with $\rho = 1.2$ and $\Gamma = 0.1$. Spatial discretization is by CDS, and a uniform grid with 20×20 CV is used. Linear equation systems in case of implicit schemes are solved using SIP solver, and the convergence error was reduced below 10^{-5}. We compute the time evolution of solution towards the steady state. Figure 6.4 shows isotherms at four time instants.

In order to investigate the accuracy of the various schemes in this case, we look at the heat flux through the isothermal wall at time $t = 0.12$. The variation of the heat flux, Q, as a function of the time step size is shown
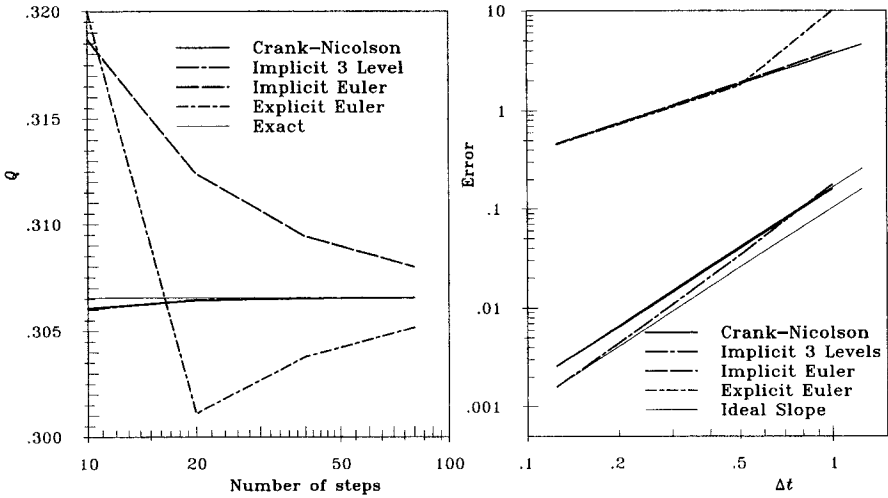
Fig. 6.5. Heat flux through the isothermal wall at $t = 0.12$ (left) and temporal discretization errors in calculated wall heat flux (right) as a function of the time step size for various schemes (spatial discretization by CDS, 20 × 20 CV uniform grid)

in Fig. 6.5. As in the previous test case, the results obtained using second order schemes change little as the number of time steps is increased, while the first order schemes are far less accurate. The explicit Euler scheme does not converge monotonically; the solution obtained with the largest time step lies on the opposite side of the time step independent value from the values obtained using smaller time steps.

For the sake of error estimation, we obtained an accurate reference solution by using very small time step ($\Delta t = 0.0003$, 400 steps to $t = 0.12$) with the Crank-Nicolson scheme. The spatial discretization was the same in all cases, so again the spatial discretization errors cancel out. By subtracting the heat flux value calculated using different schemes and time steps from the reference solution, we obtain the estimates of the temporal discretization error. These are plotted against normalized time step (normalized with the largest time step) in Fig. 6.5.

Again, the expected asymptotic convergence rates for first and second order schemes are obtained. However, the lowest error is now obtained with the second order scheme with three time levels. This is, as in the previous example, due to the dominant effect of the initial error, which turns out to be smaller when the three level scheme is started with implicit Euler method than in the Crank-Nicolson scheme. With both second order schemes the errors are much smaller than with first order schemes: the result is more accurate with second order schemes using the largest time step than with first order schemes and an eight times smaller time step!

Although we were dealing with simple unsteady problems which have a smooth transition from the initial to the steady state, we see that first order Euler schemes are very inaccurate. In both test cases the second order schemes had errors more than two orders of magnitude lower than the Euler schemes (whose errors were of the order of 1% with the smallest time step). One can expect that in transient flows even larger discrepancies can occur. Of the first order schemes only the implicit Euler method can be used when steady solutions are sought; for transient flow problems, second (or higher) order schemes are recommended.

An unsteady flow problem will be discussed in Sect. 8.11.

7. Solution of the Navier-Stokes Equations

7.1 Special Features of the Navier-Stokes Equations

In Chaps. 3, 4 and 6 we dealt with the discretization of a generic conservation equation. The discretization principles described there apply to the momentum and continuity equations (which we shall collectively call the Navier-Stokes equations). In this chapter, we shall describe how the terms in the momentum equations which differ from those in the generic conservation equation are treated.

The unsteady and advection terms in the momentum equations have the same form as in the generic conservation equation. The diffusive (viscous) terms are similar to their counterparts in the generic equation but, because the momentum equations are vector equations, these contributions become a bit more complex and their treatment needs to be considered in more detail. The momentum equations also contain a contribution from the pressure, which has no analog in the generic equation. It may be regarded either as a source term (treating the pressure gradient as body force – non-conservatively) or as a surface force (conservative treatment) but, due to the close connection of the pressure and the continuity equation, it requires special attention. Finally, the fact that the principal variable is a vector allows more freedom in the choice of a grid.

7.1.1 Discretization of Convective and Viscous Terms

The convective term in the momentum equation is non-linear; its differential and integral forms read:

$$\frac{\partial(\rho u_i u_j)}{\partial x_j} \quad \text{and} \quad \int_S \rho u_i \boldsymbol{v} \cdot \boldsymbol{n} \, \mathrm{d}S \,. \tag{7.1}$$

The treatment of the convective term in the momentum equations follows that of the convective term in the generic equation; any of the methods described in Chaps. 3 and 4 can be used.

The viscous terms in the momentum equations correspond to the diffusive term in the generic equation; their differential and integral forms are:

$$\frac{\partial \tau_{ij}}{\partial x_j} \quad \text{and} \quad \int_S (\tau_{ij} \boldsymbol{i}_j) \cdot \boldsymbol{n} \, \mathrm{d}S \,, \tag{7.2}$$

where, for a Newtonian fluid and incompressible flow:

$$\tau_{ij} = \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \,. \tag{7.3}$$

Because the momentum equations are vector equations, the viscous term is more complicated than the generic diffusive term. The part of the viscous term in the momentum equations which corresponds to the diffusive term in the generic conservation equation is

$$\frac{\partial}{\partial x_j} \left(\mu \frac{\partial u_i}{\partial x_j} \right) \quad \text{and} \quad \int_S \mu \, \mathrm{grad} \, u_i \cdot \boldsymbol{n} \, \mathrm{d}S \,. \tag{7.4}$$

This term can be discretized using any of the approaches described for the corresponding terms of the generic equation in Chaps. 3 and 4 but it is only one contribution of viscous effects to the ith component of momentum. Equations (1.28), (1.26), (1.21) and (1.17) allow us to identify the others as the contributions of the bulk viscosity (which is non-zero only in compressible flows) and a further contribution due to the spatial variability of the viscosity. For incompressible flow with constant fluid properties, these contributions disappear (thanks to the continuity equation).

The extra terms which are non-zero when the viscosity is spatially variable in an incompressible flow may be treated in the same manner as the terms (7.4):

$$\frac{\partial}{\partial x_j} \left(\mu \frac{\partial u_j}{\partial x_i} \right) \quad \text{and} \quad \int_S \left(\mu \frac{\partial u_j}{\partial x_i} \boldsymbol{i}_j \right) \cdot \boldsymbol{n} \, \mathrm{d}S \,, \tag{7.5}$$

where $\boldsymbol{n}$ is the unit outward normal to the surface of the control volume and summation on j applies. As noted above, for constant μ, this term vanishes. For this reason, this term is often treated explicitly even when implicit solution methods are used. It is argued that even when the viscosity varies this term is small compared to the term (7.4) so its treatment has only a slight impact on the rate of convergence. However, this argument applies strictly only in an integral sense; the extra term may be quite large on any one CV face.

7.1.2 Discretization of Pressure Terms and Body Forces

As noted in Chap. 1, we usually deal with the "pressure" in terms of the combination $p - \rho_0 \boldsymbol{g} \cdot \boldsymbol{r} + \mu \frac{2}{3} \mathrm{div} \, \boldsymbol{v}$. In incompressible flows, the last term is zero. One form of the momentum equations (see Eq. (1.21)) contains the gradient of this quantity which may be approximated by the FD methods

described in Chap. 3. However, as the pressure and velocity nodes on the grid may not coincide, the approximations used for their derivatives may differ.

In FV methods, the pressure term is usually treated as a surface force (conservative approach), i.e. in the equation for u_i the integral

$$-\int_S p\,\boldsymbol{i}_i \cdot \boldsymbol{n}\,\mathrm{d}S \tag{7.6}$$

is required. Methods described in Chap. 4 for the approximation of surface integrals can then be used. As we shall show below, the treatment of this term and the arrangement of variables on the grid play an important role in assuring the computational efficiency and accuracy of the numerical solution method.

Alternatively, the pressure can be treated non-conservatively, by retaining the above integral in its volumetric form:

$$-\int_\Omega \operatorname{grad} p \cdot \boldsymbol{i}_i\,\mathrm{d}\Omega\ . \tag{7.7}$$

In this case, the derivative (or, for non-orthogonal grids, all three derivatives) needs to be approximated at one or more locations within the CV. The non-conservative approach introduces a global non-conservative error; although this error tends to zero as the grid size goes to zero, it may be significant for finite grid size.

The difference between the two approaches is significant only in the FV methods. In FD methods, there is no distinction between the two versions, although one can produce both conservative and non-conservative approximations.

Other body forces, like the non-conservative ones arising when covariant or contravariant velocities are used in non-Cartesian coordinate systems are easy to treat in finite difference schemes: they are usually simple functions of one or more variables and can be evaluated using techniques described in Chap. 3. If these terms involve the unknowns, as for example, the component of the viscous term in cylindrical coordinates:

$$-2\mu\,\frac{v_r}{r^2},$$

they may be treated implicitly. This is usually done when the contribution of this term to the central coefficient A_P in the discretized equation is positive, in order to avoid destabilization of the iterative solution scheme by reducing the diagonal dominance of the matrix. Otherwise, the extra term is treated explicitly.

In FV methods, these terms are integrated over the CV volume. Usually, the mean value approach is used, so that the value at CV center is multiplied by cell volume. More elaborate schemes are possible but rarely used.

In some cases the non-conservative terms, considered as body forces, dominate the transport equation (e.g. when swirling flows are computed in polar coordinates or when flows are treated in a rotating coordinate frame, for example, in turbomachinery flows). The treatment of the non-linear source terms and the variable coupling may then become very important.

7.1.3 Conservation Properties

The Navier-Stokes equations have the property that the momentum in any control volume (microscopic or macroscopic) is changed only by flow through the surface, forces acting on the surface, and volumetric body forces. This important property is inherited by the discretized equations if the FV approach is used and the surface fluxes for adjacent control volumes are identical. If this is done, then the integral over the entire domain, being the sum of the integrals over the microscopic control volumes, reduces to a sum over the surface of the domain. Overall mass conservation follows in the same way from the continuity equation.

Energy conservation is a more complex issue. In incompressible isothermal flows, the only energy of significance is kinetic energy. When heat transfer is important, the kinetic energy is generally small compared to the thermal energy so the equation introduced to account for energy transport is a conservation equation for thermal energy. So long as the temperature dependence of the fluid properties is not significant, the thermal energy equation can be solved after solution of the momentum equations is complete. The coupling is then entirely one-way and the energy equation becomes an equation for the transport of a passive scalar, the case treated in Chaps. 3 to 6.

An equation for the kinetic energy can be derived by taking the scalar product of the momentum equation with the velocity, a procedure which mimics the derivation of the energy equation in classical mechanics. Note that, in contrast to compressible flow, for which there is a separate conservation equation for the total energy, in incompressible isothermal flows both momentum and energy conservation are consequences of the same equation; this poses the problems that are the subject of this section.

We shall be interested principally in the kinetic energy conservation equation for a macroscopic control volume, which may be either the entire considered domain or one of the small CVs used in a finite volume method. If the local kinetic energy equation obtained in the manner just described is integrated over a control volume, we obtain, after using Gauss' Theorem:

$$\frac{\partial}{\partial t}\int_{\Omega} \rho \frac{v^2}{2}\,\mathrm{d}\Omega = -\int_S \rho \frac{v^2}{2}\boldsymbol{v}\cdot\boldsymbol{n}\,\mathrm{d}S - \int_S p\boldsymbol{v}\cdot\boldsymbol{n}\,\mathrm{d}S + \int_S (\mathsf{S}\cdot\boldsymbol{v})\cdot\boldsymbol{n}\,\mathrm{d}S - \int_{\Omega}(\mathsf{S} : \operatorname{grad}\boldsymbol{v} - p\operatorname{div}\boldsymbol{v} + \rho\boldsymbol{b}\cdot\boldsymbol{v})\,\mathrm{d}\Omega\,. \qquad (7.8)$$

Here S stands for the viscous part of the stress tensor whose components are τ_{ij} defined in Eq. (1.13), i.e. $\mathsf{S} = \mathsf{T} + p\mathsf{I}$. The first term in the volume integral on the right hand side disappears if the flow is inviscid; the second is zero if the flow is incompressible; the third is zero in the absence of body forces.

Several points relating to this equation are worth mentioning.

- The first three terms on its right side are integrals over the surface of the control volume. This means that the kinetic energy in the control volume is not changed by the action of convection and/or pressure within the control volume. In the absence of viscosity, only flow of energy through the surface or work done by the forces acting at the surface of the control volume can affect the kinetic energy within it; kinetic energy is then globally conserved in this sense. This is a property that we would like to preserve in a numerical method.
- Guaranteeing global energy conservation in a numerical method is a worthwhile goal, but not an easily attained one. Because the kinetic energy equation is a consequence of the momentum equation and not a distinct conservation law, it cannot be enforced separately.
- If a numerical method is energy conservative and the net energy flux through the surface is zero then the total kinetic energy in the domain does not grow with time. If such a method is used, the velocity at every point in the domain must remain bounded, providing an important kind of numerical stability. Indeed, energy methods (which sometimes have no connection to physics) are often used to prove stability of numerical methods. Energy conservation does not say anything about the convergence or accuracy of a method. Accurate solutions may be obtained with methods that are not conservative of kinetic energy. Kinetic energy conservation is especially important in computing unsteady flows.
- Since the kinetic energy equation is a consequence of the momentum equations and not independently enforceable in a numerical method, global kinetic energy conservation must be a consequence of the *discretized* momentum equations. It is thus a property of the discretization method, but not an obvious one. To see how it might arise, we form the kinetic energy equation corresponding to the discretized momentum equations by taking the scalar product of the latter with the velocity and summing over all control volumes. We shall consider the result term-by-term.
- The pressure gradient terms are especially important, so let us look into them further. To get the pressure gradient term into the form displayed in Eq. (7.8), we used the following equality:

$$\boldsymbol{v} \cdot \operatorname{grad} p = \operatorname{div}(p\boldsymbol{v}) - p \operatorname{div} \boldsymbol{v} \; . \tag{7.9}$$

For incompressible flows, $p \operatorname{div} \boldsymbol{v} = 0$ so only the first term on the right hand side remains. As it is a divergence, its volume integral can be converted to a surface integral. As already noted, this means that the pressure influences the overall kinetic energy budget only by its action at the surface. We

would like the discretization to retain this property. Let us see how this might happen.

If $G_i p$ represents the numerical approximation to the ith component of the pressure gradient, then, when the discretized u_i-momentum equation is multiplied by u_i, the pressure gradient term gives a contribution $\sum u_i\, G_i p\, \Delta\Omega$. Energy conservation requires that this contribution be equal to (cf. Eq. (7.9)):

$$\sum_{i=1}^{N} u_i\, G_i p\, \Delta\Omega = \sum_{S_b} p v_n\, \Delta S - \sum_{N} p\, D_i u_i\, \Delta\Omega \ , \tag{7.10}$$

where the subscript N indicates that the sum is over all CVs (grid nodes), S_b is the boundary of the solution domain, v_n is the velocity component normal to the boundary and $D_i u_i$ is the discretized velocity divergence used in the continuity equation. If this is so, $D_i u_i = 0$ at each node, so the second term in the above equation is zero. The equality of the left and right hand sides can then be ensured only if G_i and D_i are compatible in the following sense:

$$\sum_{i=1}^{N} (u_i\, G_i p + p\, D_i u_i)\, \Delta\Omega = \text{ surface terms} \ . \tag{7.11}$$

This states that the approximation of the pressure gradient and the divergence of the velocity must be compatible if kinetic energy conservation is to hold. Once either approximation is chosen, the freedom to choose the other is lost.

To make this more concrete, assume that the pressure gradient is approximated with backward differences and the divergence operator with forward differences (the usual choice on a staggered grid). The one-dimensional version of Eq. (7.11) on a uniform grid then reads:

$$\sum_{i=1}^{N} [(p_i - p_{i-1}) u_i + (u_{i+1} - u_i) p_i] = u_{N+1} p_N - u_1 p_0 \tag{7.12}$$

The only two terms that remain when the sum is taken are the "surface terms" on the right hand side. The two operators are therefore compatible in the above sense. Conversely, if forward differences were used for the pressure gradient, the continuity equation would need to use backward differences. If central differences are used for one, they are required for the other.

The requirement that only boundary terms remain when the sum over all CVs (grid nodes) is taken applies to the other two conservative terms, the convective and viscous stress terms. Satisfaction of this requirement is not easy in any case and is especially difficult for arbitrary and unstructured

meshes. If a method is not energy conservative on uniform regular grids, it will certainly not be so on more complicated ones. On the other hand, a method which is conservative on uniform grids might be nearly so on complex grids.

- A Poisson equation is often used to compute the pressure. As we shall see, it is derived by taking the divergence of the momentum equation. The Laplacian operator in the Poisson equation is thus the product of the divergence operator in the continuity equation and the gradient operator in the momentum equation. The approximation of the Poisson equation cannot be selected independently; it must be consistent with the divergence and gradient operators if mass conservation is to obtain. Energy conservation adds the further requirement that the divergence and gradient approximations be consistent in the sense defined above.
- For an incompressible flow without body forces, the only remaining volume integral is the viscous term. For a Newtonian fluid, this term becomes:

$$-\int_{\Omega} \tau_{ij} \frac{\partial u_j}{\partial x_i} \, \mathrm{d}\Omega \, .$$

Inspection reveals that the integrand is a sum of squares so this term is always negative (or zero). It represents the irreversible (in the thermodynamic sense) conversion of kinetic energy of the flow into internal energy of the fluid and is called viscous dissipation. As incompressible flows are usually low speed flows, the addition to the internal energy is rarely significant but the loss of kinetic energy is often quite important to the flow. In compressible flows, the energy transfer is often important to both sides.

- The time differencing method can destroy the energy conservation property. In addition to the requirements on the spatial discretization mentioned above, the approximation of the time derivatives should be properly chosen. The Crank-Nicolson scheme is a particularly good choice. In it, the time derivatives in the momentum equations are approximated by:

$$\frac{\rho \, \Delta\Omega}{\Delta t} \left(u_i^{n+1} - u_i^n \right) .$$

If we take the scalar product of this term with $u_i^{n+1/2}$, which in the Crank-Nicolson scheme is approximated by $(u_i^{n+1} + u_i^n)/2$, the result is the change in the kinetic energy:

$$\frac{\rho \, \Delta\Omega}{\Delta t} \left[\left(\frac{v^2}{2} \right)^{n+1} - \left(\frac{v^2}{2} \right)^{n} \right] .$$

where $v^2 = u_i u_i$ (summation implied). With proper choices of the approximations to the other terms, the Crank-Nicolson scheme is energy conservative.

The fact that momentum and energy conservation are both governed by the same equation makes construction of numerical approximations that conserve both properties difficult. As already noted, kinetic energy conservation cannot be enforced independently. If the momentum equations are written in strong conservation form and a finite volume method is used then global momentum conservation is usually assured. The construction of energy conservative methods is a hit or miss affair. One selects a method and determines whether it is conservative or not; if not, adjustments are made until conservation is achieved.

An alternative method of guaranteeing kinetic energy conservation is to use a different form of the momentum equations. For example, one could use the following equation for incompressible flows:

$$\frac{\partial u_i}{\partial t} + \epsilon_{ijk} u_j \omega_k = \frac{\partial \left(\frac{p}{\rho} + \frac{1}{2} u_j u_j \right)}{\partial x_i} + \nu \frac{\partial^2 u_i}{\partial x_j \partial x_j} , \tag{7.13}$$

where ϵ_{ijk} is the Levi-Civita symbol (it is $+1$ if $\{ijk\} = \{123\}$ or an even permutation of it, it is -1 if $\{ijk\}$ is an odd permutation of $\{123\}$ such as $\{321\}$ and zero otherwise). $\boldsymbol{\omega}$ is the vorticity defined by Eq. (7.64). Energy conservation follows from this form of the momentum equation by symmetry; when the equation is multiplied by u_i, the second term on the left hand side is identically zero as a consequence of the antisymmetry property of ϵ_{ijk}. However, because this is not a conservative form of the momentum equation, construction of a momentum conserving method requires care.

Kinetic energy conservation is of particular importance in computing complex unsteady flows. Examples include the simulation of global weather patterns and simulations of turbulent flows. Lack of guaranteed energy conservation in these simulations often leads to growth of the kinetic energy and instability. For steady flows, energy conservation is less important but it does prevent certain types of misbehavior by the iterative solution method.

Kinetic energy is not the only quantity whose conservation is desirable but cannot be independently enforced: angular momentum is another such quantity. Flows in rotating machinery, internal combustion engines and many other devices exhibit pronounced rotation or swirl. If the numerical scheme does not conserve global angular momentum, the calculation is likely to get into trouble. Central difference schemes are generally much better than upwind schemes with respect to angular momentum conservation.

7.2 Choice of Variable Arrangement on the Grid

Now let us turn to the discretizations. The first issue is to select the points in the domain at which the values of the unknown dependent variables are to be computed. There is more to this than one might think. Basic features

of numerical grids were outlined in Chap. 2. There are, however, many variants of the distribution of computational points within the solution domain. The basic arrangements associated with the FD and FV discretization methods were shown in Figs. 3.1 and 4.1. These arrangements may become more complicated when coupled equations for vector fields (like the Navier-Stokes equations) are being solved. These issues are discussed below.

7.2.1 Colocated Arrangement

The obvious choice is to store all the variables at the same set of grid points and to use the same control volumes for all variables; such a grid is called *colocated*, see Fig. 7.1. Since many of the terms in each of the equations are essentially identical, the number of coefficients that must be computed and stored is minimized and the programming is simplified by this choice. Furthermore, when multigrid procedures are used, the same restriction and prolongation operators for transfer of information between the various grids can be used for all variables.

The colocated arrangement also has significant advantages in complicated solution domains, especially when the boundaries have slope discontinuities or the boundary conditions are discontinuous. A set of control volumes can be designed to fit the boundary including the discontinuity. Other arrangements of the variables lead to some of the variables being located at singularities of the grid, which may lead to singularities in the discretized equations.
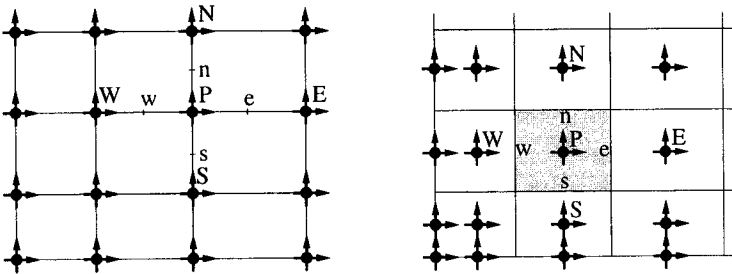
Fig. 7.1. Colocated arrangement of velocity components and pressure on a FD (left) and FV (right) grid

The colocated arrangement was out of favor for a long time for incompressible flow computation due to the difficulties with pressure-velocity coupling and the occurrence of oscillations in the pressure. From the time the staggered grid was introduced in the mid-1960s until the early 1980s, the colocated arrangement was hardly used. Then, use of non-orthogonal grids became more commonplace as problems in complex geometries began to be tackled. The staggered approach can be used in generalized coordinates only

when contravariant (or other grid-oriented) components of vectors and tensors are the working variables. This complicates the equations by introducing curvature terms that are difficult to treat numerically, and may create non-conservative errors when the grid is not smooth as will be shown in Chap. 8. When improved pressure-velocity coupling algorithms were developed in the 1980's, the popularity of the colocated arrangement began to rise. The advantages will be demonstrated below.

7.2.2 Staggered Arrangements

There is no need for all variables to share the same grid; a different arrangement may turn out to be advantageous. In Cartesian coordinates, the staggered arrangement introduced by Harlow and Welsh (1965) offers several advantages over the colocated arrangement. This arrangement is shown in Fig. 7.2. Several terms that require interpolation with the colocated arrangement, can be calculated (to a second-order approximation) without interpolation. This can be seen from the x-momentum CV shown in Fig. 7.4. Both the pressure and diffusion terms are very naturally approximated by central difference approximations without interpolation, since the pressure nodes lie at CV face centers and the velocity derivatives needed for the diffusive terms are readily computed at the CV faces. Also, evaluation of mass fluxes in the continuity equation on the faces of a pressure CV is straightforward. Details are presented below.
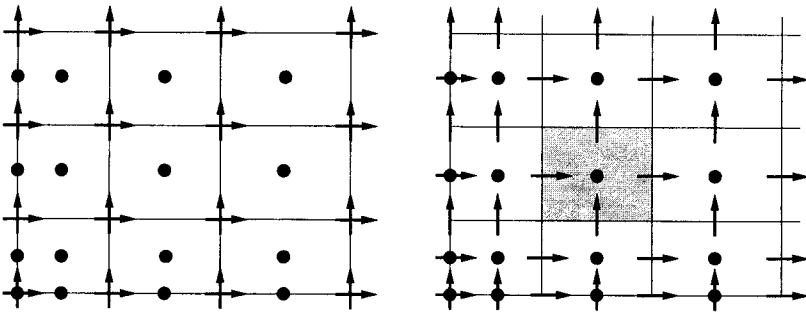
Fig. 7.2. Fully staggered arrangement of velocity components and pressure (right) and a partially staggered arrangement (left) on a FV grid

Perhaps the biggest advantage of the staggered arrangement is the strong coupling between the velocities and the pressure. This helps to avoid some types of convergence problems and oscillations in pressure and velocity fields, an issue that will be further discussed later.

The numerical approximation on a staggered grid is also conservative of kinetic energy which has advantages that were discussed earlier. The proof of this statement is straight-forward but lengthy and will not be given here.

Other staggering methods have been suggested, for example, the partially staggered ALE (Arbitrary Lagrangian-Eulerian) method (Hirt et al., 1974), in which both velocity components are stored at the corners of the pressure CVs, see Fig. 7.2. This variant has some advantages when the grid is non-orthogonal, an important one being that the pressure at the boundary need not be specified. However, it also has drawbacks, notably the possibility of producing oscillatory pressure or velocity fields.

Other arrangements have not gained wide popularity and will not be further discussed here.

7.3 Calculation of the Pressure

Solution of the Navier-Stokes equations is complicated by the lack of an independent equation for the pressure, whose gradient contributes to each of the three momentum equations. Furthermore, the continuity equation does not have a dominant variable in incompressible flows. Mass conservation is a kinematic constraint on the velocity field rather than a dynamic equation. One way out of this difficulty is to construct the pressure field so as to guarantee satisfaction of the continuity equation. This may seem a bit strange at first, but we shall show below that it is possible. Note that the absolute pressure is of no significance in an incompressible flow; only the gradient of the pressure (pressure difference) affects the flow.

In compressible flows the continuity equation can be used to determine the density and the pressure is calculated from an equation of state. This approach is not appropriate for incompressible or low Mach number flows.

Within this section we present the basic philosophy behind some of the most popular methods of pressure-velocity coupling. Section 7.5 presents a full set of discretized equations which form the basis for writing a computer code.

7.3.1 The Pressure Equation and its Solution

The momentum equations clearly determine the respective velocity components so their roles are clearly defined. This leaves the continuity equation, which does not contain the pressure, to determine the pressure. How can this be done? The most common method is based on combining the two equations.

The form of the continuity equation suggests that we take the divergence of the momentum equation (1.15). The continuity equation can be used to simplify the resulting equation, leaving a Poisson equation for the pressure:

$$\operatorname{div}(\operatorname{grad} p) = -\operatorname{div}\left[\operatorname{div}(\rho \boldsymbol{v}\boldsymbol{v} - \mathsf{S}) - \rho \boldsymbol{b} + \frac{\partial(\rho \boldsymbol{v})}{\partial t}\right] . \tag{7.14}$$

In Cartesian coordinates this equation reads:

$$\frac{\partial}{\partial x_i}\left(\frac{\partial p}{\partial x_i}\right) = -\frac{\partial}{\partial x_i}\left[\frac{\partial}{\partial x_j}\left(\rho u_i u_j - \tau_{ij}\right)\right] + \frac{\partial(\rho b_i)}{\partial x_i} + \frac{\partial^2 \rho}{\partial t^2} . \tag{7.15}$$

For the case of constant density and viscosity, this equation simplifies further; the viscous and unsteady terms disappear by virtue of the continuity equation leaving:

$$\frac{\partial}{\partial x_i}\left(\frac{\partial p}{\partial x_i}\right) = -\frac{\partial}{\partial x_i}\left[\frac{\partial(\rho u_i u_j)}{\partial x_j}\right] . \tag{7.16}$$

The pressure equation can be solved by one of the numerical methods for elliptic equations described in Chaps. 3 and 4. It is important to note that the right hand side of the pressure equation is a sum of derivatives of terms in the momentum equations; these must be approximated in a manner consistent with their treatment in the equations they are derived from.

It is also important to note that the Laplacian operator in the pressure equation is the product of the divergence operator originating from the continuity equation and the gradient operator that comes from the momentum equations. In a numerical approximation, it is essential that the consistency of these operators be maintained i.e. the approximation of the Poisson equations must be defined as the product of the divergence and gradient approximations used in the basic equations. Violation of this constraint leads to lack of satisfaction of the continuity equation. To emphasize the importance of this issue, the two derivatives of the pressure in the above equations were separated: the outer derivative stems from the continuity equation while the inner derivative arises from the momentum equations. The outer and inner derivatives may be discretized using different schemes – they have to be those used in the momentum and continuity equations.

A pressure equation of this kind is used to calculate the pressure in both explicit and implicit solution methods. To maintain consistency among the approximations used, it is best to derive the equation for the pressure from the discretized momentum and continuity equations rather than by approximating the Poisson equation. The pressure equation can also be used to calculate the pressure from a velocity field obtained by solving vorticity/streamfunction equations, see Sect. 7.4.2.

7.3.2 A Simple Explicit Time Advance Scheme

Before considering commonly used methods for solving the steady state Navier-Stokes equations, let us look at a method for the unsteady equations that illustrates how the numerical Poisson equation for the pressure is constructed and the role it plays in enforcing continuity. The choice of the approximations to the spatial derivatives is not important here so the semi-discretized (discrete in space but not time) momentum equations are written symbolically as:

$$\frac{\partial(\rho u_i)}{\partial t} = -\frac{\delta(\rho u_i u_j)}{\delta x_j} - \frac{\delta p}{\delta x_i} + \frac{\delta \tau_{ij}}{\delta x_j} = H_i - \frac{\delta p}{\delta x_i} \, , \tag{7.17}$$

where $\delta/\delta x$ represents a discretized spatial derivative (which could represent a different approximation in each term) and H_i is shorthand notation for the advective and viscous terms whose treatment is of no importance here.

For simplicity, assume that we wish to solve Eq. (7.17) with the explicit Euler method for time advancement. We then have:

$$(\rho u_i)^{n+1} - (\rho u_i)^n = \Delta t \left(H_i^n - \frac{\delta p^n}{\delta x_i} \right) \, . \tag{7.18}$$

To apply this method, the velocity at time step n is used to compute H_i^n and, if the pressure is available, $\delta p^n/\delta x_i$ may also be computed. This gives an estimate of ρu_i at the new time step $n+1$. In general, this velocity field does not satisfy the continuity equation:

$$\frac{\delta(\rho u_i)^{n+1}}{\delta x_i} = 0 \, . \tag{7.19}$$

We have stated an interest in incompressible flows, but these include flows with variable density; this is emphasized by including the density. To see how continuity may be enforced, let us take the numerical divergence (using the numerical operators used to approximate the continuity equation) of Eq. (7.18). The result is:

$$\frac{\delta(\rho u_i)^{n+1}}{\delta x_i} - \frac{\delta(\rho u_i)^n}{\delta x_i} = \Delta t \left[\frac{\delta}{\delta x_i} \left(H_i^n - \frac{\delta p^n}{\delta x_i} \right) \right] \, . \tag{7.20}$$

The first term is the divergence of the new velocity field, which we want to be zero. The second term is zero if continuity was enforced at time step n; we shall assume that this is the case but, if it is not, this term should be left in the equation. Retaining this term is necessary when an iterative method is used to solve the Poisson equation for the pressure and the iterative process is not converged completely. Similarly, the divergence of the viscous component of H_i should be zero for constant ρ, but a non-zero value is easily accounted for. Taking all this into account, the result is the discrete Poisson equation for the pressure p^n:

$$\frac{\delta}{\delta x_i} \left(\frac{\delta p^n}{\delta x_i} \right) = \frac{\delta H_i^n}{\delta x_i} \, . \tag{7.21}$$

Note that the operator $\delta/\delta x_i$ outside the parentheses is the divergence operator inherited from the continuity equation, while $\delta p/\delta x_i$ is the pressure gradient from the momentum equations. If the pressure p^n satisfies this discrete Poisson equation, the velocity field at time step $n+1$ will be divergence free (in terms of the discrete divergence operator). Note that the time step

to which this pressure belongs is arbitrary. If the pressure gradient term had been treated implicitly, we would have p^{n+1} in place of p^n but everything else would remain unchanged.

This provides the following algorithm for time-advancing the Navier-Stokes equations:

- Start with a velocity field u_i^n at time t_n which is assumed divergence free. (As noted, if it is not divergence free this can be corrected.)
- Compute the combination, H_i^n, of the advective and viscous terms and its divergence (both need to be retained for later use).
- Solve the Poisson equation for the pressure p^n.
- Compute the velocity field at the new time step. It will be divergence free.
- The stage is now set for the next time step.

Methods similar to this are commonly used to solve the Navier-Stokes equations when an accurate time history of the flow is required. The principal differences in practice are that time advancement methods more accurate than the first order Euler method are usually used and that some of the terms may be treated implicitly. Some of these methods will be described later.

We have shown how solving the Poisson equation for the pressure can assure that the velocity field satisfies the continuity equation i.e. that it is divergence free. This idea runs through many of the methods used to solve both the steady and unsteady Navier-Stokes equations. We shall now study some of the more commonly used methods for solving the steady Navier-Stokes equations.

7.3.3 A Simple Implicit Time Advance Method

To see what additional difficulties arise when an implicit method is used to solve the Navier-Stokes equations, let us construct such a method. Since we are interested in illuminating certain issues, let us use a scheme based on the the simplest implicit method, the backward or implicit Euler method. If we apply this method to Eq. (7.17), we have:

$$(\rho u_i)^{n+1} - (\rho u_i)^n = \Delta t \left(-\frac{\delta(\rho u_i u_j)^{n+1}}{\delta x_j} - \frac{\delta p^{n+1}}{\delta x_i} + \frac{\delta \tau_{ij}^{n+1}}{\delta x_j} \right) . \quad (7.22)$$

We see immediately that there are difficulties that were not present in the explicit method described in the preceding section. Let us consider these one at a time.

First, there is a problem with the pressure. The divergence of the velocity field at the new time step must be zero. This can be accomplished in much the same way as in the explicit method. We take the divergence of Eq. (7.22), assume that the velocity field at time step n is divergence free (this can be

corrected for if necessary) and demand that the divergence at the new time step $n+1$ also be zero. This leads to the Poisson equation for the pressure:

$$\frac{\delta}{\delta x_i}\left(\frac{\delta p^{n+1}}{\delta x_i}\right) = \frac{\delta}{\delta x_i}\left(\frac{-\delta(\rho u_i u_j)^{n+1}}{\delta x_j}\right) . \tag{7.23}$$

The problem is that the term on the right hand side cannot be computed until the computation of the velocity field at time $n+1$ is completed and vice versa. As a result, the Poisson equation and the momentum equations have to be solved simultaneously. That can only be done with some type of iterative procedure.

Next, even if the pressure were known, Eqs. (7.22) are a large system of non-linear equations which must be solved for the velocity field. The structure of this system of equations is essentially the same as the structure of the matrix of the finite-differenced Laplace equation so solving them is far from a trivial matter. If one wishes to solve them accurately, the best procedure is to adopt the converged results from the preceding time step as the initial guess for the new velocity field and then converge to the solution at the new time step using the Newton–Raphson iteration method or a secant method designed for systems.

An alternative way of dealing with the non-linearity is constructed by linearizing the equations about the result at the preceding time step. If we write:

$$u_i^{n+1} = u_i^n + \Delta u_i , \tag{7.24}$$

then the non-linear term in Eqs. (7.22) can be expressed as:

$$u_i^{n+1} u_j^{n+1} = u_i^n u_j^n + u_i^n \Delta u_j + u_j^n \Delta u_i + \Delta u_i \Delta u_j . \tag{7.25}$$

We expect that, at least for small Δt, $\Delta u_i \sim \Delta t\, \partial u_i/\partial t$, so the last term in this equation is second order in Δt and is smaller in magnitude than the error made in the time discretization. It can therefore be neglected. If we use a second order method in time, such as the Crank-Nicolson scheme, this term would be of the same order as the spatial discretization error and we would still be justified in neglecting it.

We can then write Eq. (7.22) as:

$$\rho \Delta u_i = \Delta t \left(-\frac{\delta(\rho u_i u_j)^n}{\delta x_j} - \frac{\delta(\rho u_i^n \Delta u_j)}{\delta x_j} - \frac{\delta(\rho \Delta u_i u_j^n)}{\delta x_j} - \right.$$
$$\left. \frac{\delta p^n}{\delta x_i} - \frac{\delta \Delta p}{\delta x_i} + \frac{\delta \tau_{ij}^n}{\delta x_j} + \frac{\delta \Delta \tau_{ij}}{\delta x_j} \right) . \tag{7.26}$$

This method takes advantage of the fact that the non-linearity is quadratic and removes most of the difficulty arising from it. However, we still need to solve a large system of linear equations with the structure discussed above. Direct solution of such a system is too expensive to consider so the solution

needs to be found iteratively. An interesting possibility is to use the alternating direction implicit (ADI) method to split the equations into a series of one dimensional problems, each of which is block tridiagonal. The solution at the new time step can then be found with sufficient accuracy with just one iteration (one set of block tridiagonal solutions in each direction).

So a reasonable strategy is to use the local linearization based on Eq. (7.24) and update the equations by the ADI method using the old pressure gradient. We can then correct the velocity field using the following scheme.

- Call the velocity field computed by updating the momentum equations with the old pressure gradient u_i^*. It does not satisfy the continuity equation.
- Solve a Poisson equation for the pressure correction:

$$\frac{\delta}{\delta x_i}\left(\frac{\delta \Delta p}{\delta x_i}\right) = \frac{1}{\Delta t}\frac{\delta(\rho u_i^*)}{\delta x_i} . \tag{7.27}$$

- Update the velocity:

$$u_i^{n+1} = u_i^* - \frac{\Delta t}{\rho}\frac{\delta \Delta p}{\delta x_i} , \tag{7.28}$$

  which does satisfy continuity.

With these tricks, the method suggested here is about twice as expensive as the explicit method per time step.

The method described above is designed to produce an accurate solution of an unsteady problem. In problems of that kind, the required accuracy in time usually sets the time step, which will be rather small. Because they allow large time steps to be used without instability, implicit methods are often used to solve steady state problems. The idea is to compute in time until a steady solution is obtained. In this type of calculation, the error made in linearizing the problem is no longer negligible and the type of method described here may not be the best choice. Methods designed for solving steady state problems are given in the next section. They introduce other means of getting around the problems we encountered here.

7.3.4 Implicit Pressure-Correction Methods

As noted in Chap. 6, many methods for steady problems can be regarded as solving an unsteady problem until a steady state is reached. The principal difference is that, when solving an unsteady problem, the time step is chosen so that an accurate history is obtained while, when a steady solution is sought, large time steps are used to try to reach the steady state quickly. Implicit methods are preferred for steady and slow-transient flows, because they have less stringent time step restrictions than explicit schemes (they may not have any).

Many solution methods for steady incompressible flows are of the latter type; some of the most popular ones can be regarded as variations on the method of the preceding section. They use a pressure (or pressure-correction) equation to enforce mass conservation at each time step or, in the language preferred for steady solvers, each outer iteration. We now look at some of these methods.

If an implicit method is used to advance the momentum equations in time, the discretized equations for the velocities at the new time step are non-linear. If the pressure gradient term is not included in the source term, these may be written:

$$A_{\mathrm{P}}^{u_i} u_{i,\mathrm{P}}^{n+1} + \sum_l A_l^{u_i} u_{i,l}^{n+1} = Q_{u_i}^{n+1} - \left(\frac{\delta p^{n+1}}{\delta x_i} \right)_{\mathrm{P}} . \qquad (7.29)$$

As always, P is the index of an arbitrary velocity node, and index l denotes the neighbor points that appear in the discretized momentum equation. The source term Q contains all of the terms that may be explicitly computed in terms of u_i^n as well as any body force or other linearized terms that may depend on the u_i^{n+1} or other variables at the new time level (like temperature) – hence the superscript $n+1$. The pressure term is written in symbolic difference form to emphasize the independence of the solution method from the discretization approximation for the spatial derivatives. The discretizations of the spatial derivatives may be of any order or any type described in Chap. 3.

Due to the non-linearity and coupling of the underlying differential equations, Eqs. (7.29) cannot be solved directly as the coefficients A and, possibly, the source term, depend on the unknown solution u_i^{n+1}. Iterative solution is the only choice; some approaches were described in Chap. 5. If we are computing an unsteady flow and time accuracy is required, iteration must be continued within each time step until the entire system of non-linear equations is satisfied to within a narrow tolerance. For steady flows, the tolerance can be much more generous; one can then either take an infinite time step and iterate until the steady non-linear equations are satisfied, or march in time without requiring full satisfaction of the non-linear equations at each time step.

The iterations within one time step, in which the coefficient and source matrices are updated, are called *outer iterations* to distinguish them from the *inner iterations* performed on linear systems with fixed coefficients. On each outer iteration, the equations solved are:

$$A_{\mathrm{P}}^{u_i} u_{i,\mathrm{P}}^{m*} + \sum_l A_l^{u_i} u_{i,l}^{m*} = Q_{u_i}^{m-1} - \left(\frac{\delta p^{m-1}}{\delta x_i} \right)_{\mathrm{P}} . \qquad (7.30)$$

We dropped the time step index $n+1$ and introduced an outer iteration counter m; u_i^m thus represents the current estimate of the solution u_i^{n+1}. At

the beginning of each outer iteration, the terms on the right hand side of Eq. (7.30) are evaluated using the variables at the preceding outer iteration.

The momentum equations are usually solved sequentially i.e. the set of algebraic equations for each component of the momentum is solved in turn, treating the grid point values of its dominant velocity component as the sole set of unknowns. Since the pressure used in these iterations was obtained from the previous outer iteration or time step, the velocities computed from Eqs. (7.30) do not normally satisfy the discretized continuity equation. To enforce the continuity condition, the velocities need to be corrected; this requires modification of the pressure field; the manner of doing this is described next.

The velocity at node P, obtained by solving the linearized momentum equations (7.30), can be formally expressed as:

$$u_{i,\mathrm{P}}^{m*} = \frac{Q_{u_i}^{m-1} - \sum_l A_l^{u_i} u_{i,l}^{m*}}{A_\mathrm{P}^{u_i}} - \frac{1}{A_\mathrm{P}^{u_i}} \left(\frac{\delta p^{m-1}}{\delta x_i} \right)_\mathrm{P} . \tag{7.31}$$

As already stated, these velocities do not satisfy the continuity equation, so u_{iP}^{m*} is not the final value of the velocity for iteration m; it is a predicted value, which is why it carries an asterisk (*). The corrected final values should satisfy the continuity equation. For convenience, the first term on the right hand side of the above equations is called $\tilde{u}_{i,\mathrm{P}}^{m*}$:

$$u_{i,\mathrm{P}}^{m*} = \tilde{u}_{i,\mathrm{P}}^{m*} - \frac{1}{A_\mathrm{P}^{u_i}} \left(\frac{\delta p^{m-1}}{\delta x_i} \right)_\mathrm{P} . \tag{7.32}$$

The velocity field $\tilde{u}_i^{m*}$ can be thought of as one from which the contribution of the pressure gradient has been removed. Because the method is implicit, this is not the velocity that would be obtained by dropping the pressure gradient entirely from Eq. (7.30).

The next task is to correct the velocities so that they satisfy the continuity equation:

$$\frac{\delta(\rho u_i^m)}{\delta x_i} = 0 , \tag{7.33}$$

which can be achieved by correcting the pressure field. The corrected velocities and pressure are linked by (see Eq. (7.32)):

$$u_{i,\mathrm{P}}^{m} = \tilde{u}_{i,\mathrm{P}}^{m*} - \frac{1}{A_\mathrm{P}^{u_i}} \left(\frac{\delta p^{m}}{\delta x_i} \right)_\mathrm{P} . \tag{7.34}$$

Continuity is now enforced by inserting this expression for u_i^m into the continuity equation (7.33), to yield a discrete Poisson equation for the pressure:

$$\frac{\delta}{\delta x_i} \left[\frac{\rho}{A_\mathrm{P}^{u_i}} \left(\frac{\delta p^m}{\delta x_i} \right) \right]_\mathrm{P} = \left[\frac{\delta(\rho \tilde{u}_i^{m*})}{\delta x_i} \right]_\mathrm{P} . \tag{7.35}$$

As noted earlier, the derivatives of the pressure inside the brackets must be discretized in the same way they are discretized in the momentum equations; the outer derivatives, which come from the continuity equation, must be approximated in the way they are discretized in the continuity equation.

After solving the Poisson equation for the pressure, (7.35), the final velocity field at the new iteration, u_i^m, is calculated from Eq. (7.34). At this point, we have a velocity field which satisfies the continuity condition, but the velocity and pressure fields do not satisfy the momentum equations (7.30). We begin another outer iteration and the process is continued until a velocity field which satisfies both the momentum and continuity equations is obtained.

This method is essentially a variation on the one presented in the preceding section. Methods of this kind, which first construct velocity field that does not satisfy the continuity equation and then correct it by subtracting something (usually a pressure gradient) are known as *projection methods.* The name is derived from the concept that the divergence-producing part of the field is projected out.

In one of the most common methods of this type, a pressure-correction is used instead of the actual pressure. The velocities computed from the linearized momentum equations and the pressure p^{m-1} are taken as provisional values to which a small correction must be added:

$$u_i^m = u_i^{m*} + u' \quad \text{and} \quad p^m = p^{m-1} + p' \tag{7.36}$$

If these are substituted into the momentum equations (7.30), we obtain the relation between the velocity and pressure corrections:

$$u'_{i,\mathrm{P}} = \tilde{u}'_{i,\mathrm{P}} - \frac{1}{A_\mathrm{P}^{u_i}} \left(\frac{\delta p'}{\delta x_i} \right)_\mathrm{P} , \tag{7.37}$$

where $\tilde{u}'_i$ is defined by (see Eq. (7.31)):

$$\tilde{u}'_{i,\mathrm{P}} = - \frac{\sum_l A_l^{u_i} u'_{i,l}}{A_\mathrm{P}^{u_i}} . \tag{7.38}$$

Application of the discretized continuity equation (7.33) to corrected velocities and use of expression (7.37) produces the following pressure-correction equation:

$$\frac{\delta}{\delta x_i} \left[\frac{\rho}{A_\mathrm{P}^{u_i}} \left(\frac{\delta p'}{\delta x_i} \right) \right]_\mathrm{P} = \left[\frac{\delta (\rho u_i^{m*})}{\delta x_i} \right]_\mathrm{P} + \left[\frac{\delta (\rho \tilde{u}'_i)}{\delta x_i} \right]_\mathrm{P} . \tag{7.39}$$

The velocity corrections $\tilde{u}'_i$ are unknown at this point, so it is common practice to neglect them. This is hard to justify and is probably the major reason why the resulting method does not converge very rapidly.

Alternative methods that are less brutal to the velocity correction will be described below. In the present method, once the pressure correction has

been solved for, the velocities are updated using Eqs. (7.36) and (7.37). This is known as the SIMPLE algorithm (Caretto et al., 1972), an acronym whose origin will not be detailed. We shall discuss its properties below.

A more gentle way of treating the last term in the pressure-correction equation (7.39) is to approximate it rather than neglecting it. One could approximate the velocity correction u_i' at any node by a weighted mean of the neighbor values, for example,

$$u_{i,\mathrm{P}}' \approx \frac{\sum_l A_l^{u_i} u_{i,l}'}{\sum_l A_l^{u_i}} \ . \tag{7.40}$$

This allows us to approximate $\tilde{u}_{i,\mathrm{P}}'$ from Eq. (7.38) as

$$\tilde{u}_{i,\mathrm{P}}' \approx -u_{i,\mathrm{P}}' \frac{\sum_l A_l^{u_i}}{A_\mathrm{P}^{u_i}} \ , \tag{7.41}$$

which, when inserted in Eq. (7.37), leads to the following approximate relation between u_i' and p':

$$u_{i,\mathrm{P}}' = -\frac{1}{A_\mathrm{P}^{u_i} + \sum_l A_l^{u_i}} \left(\frac{\delta p'}{\delta x_i} \right)_\mathrm{P} \ . \tag{7.42}$$

With this approximation the coefficient $A_\mathrm{P}^{u_i}$ in Eq. (7.39) is replaced by $A_\mathrm{P}^{u_i} + \sum_l A_l^{u_i}$ and the last term disappears. This is known as the SIMPLEC algorithm (van Doormal and Raithby, 1984).

Still another method of this general type is derived by neglecting $\tilde{u}_i'$ in the first correction step as in the SIMPLE method but following the correction with another corrector step. The second correction to the velocity u'' is defined by (see Eq. (7.37)):

$$u_{i,\mathrm{P}}'' = \tilde{u}_{i,\mathrm{P}}' - \frac{1}{A_\mathrm{P}^{u_i}} \left(\frac{\delta p''}{\delta x_i} \right)_\mathrm{P} \ ; \tag{7.43}$$

where $\tilde{u}_i'$ is calculated from Eq. (7.38) after u_i' has been calculated from Eq. (7.37) with $\tilde{u}_i'$ neglected. Application of the discretized continuity equation (7.33) to corrected velocities leads to the second pressure-correction equation:

$$\frac{\delta}{\delta x_i} \left[\frac{\rho}{A_\mathrm{P}^{u_i}} \left(\frac{\delta p''}{\delta x_i} \right) \right]_\mathrm{P} = \left[\frac{\delta (\rho \tilde{u}_i')}{\delta x_i} \right]_\mathrm{P} \ . \tag{7.44}$$

Note that the coefficients on the left hand side are the same as in Eq. (7.39), which can be exploited (a factorization of the matrix may be stored and reused). Still further corrector steps can be constructed in the same way, but this is seldom done. This procedure is essentially an iterative method for solving Eq. (7.39) with the last term treated explicitly; it is known as the PISO algorithm (Issa, 1986).

Another similar method of this kind was proposed by Patankar (1980) and is called SIMPLER. In it, the pressure-correction equation (7.39) is solved first with the last term neglected as in SIMPLE. The pressure correction so obtained is used only to correct the velocity field so that it satisfies continuity i.e. to obtain u_i^m. The new pressure field is calculated from the pressure equation (7.35) using $\tilde{u}_i^m$ instead of $\tilde{u}_i^{m*}$. This is possible because u_i^m is now available.

As already noted, due to the neglect of $\tilde{u}_i'$ in Eq. (7.39) (which is equivalent to neglecting it in Eq. (7.37)), the SIMPLE algorithm does not converge rapidly. Its performance depends greatly on the size of time step, or – for steady flows – on the value of the under-relaxation parameter used in the momentum equations. It has been found by trial and error that convergence can be improved if one adds only a portion of p' to p^{m-1}, i.e. if one takes

$$p^m = p^{m-1} + \alpha_p p' \tag{7.45}$$

after the pressure-correction equation is solved, where $0 \leq \alpha_p \leq 1$. SIMPLEC, SIMPLER and PISO do not need under-relaxation of the pressure correction.

One can derive an optimum relation between the under-relaxation factors for velocities and pressure by the following argument[1].

The velocities in the SIMPLE method are corrected by

$$u'_{i,\mathrm{P}} = -\frac{1}{A_\mathrm{P}^{u_i}} \left(\frac{\delta p'}{\delta x_i}\right)_\mathrm{P} , \tag{7.46}$$

i.e., $\tilde{u}'_{i,\mathrm{P}}$ is neglected. To make up for this crudeness, we may now go back to the momentum equations (7.31) and look for pressure which will satisfy these equations when u_i^{m*} is replaced by corrected velocities u_i^m, which now satisfy the continuity equation (this is the path that leads to the pressure equation in SIMPLER). By assuming that the final pressure correction is $\alpha_p p'$, we arrive at the following equation:

$$u'_{i,\mathrm{P}} = \tilde{u}'_{i,\mathrm{P}} - \alpha_p \frac{1}{A_\mathrm{P}^{u_i}} \left(\frac{\delta p'}{\delta x_i}\right)_\mathrm{P} . \tag{7.47}$$

By making use of Eq. (7.46), we arrive at the following expression for α_p:

$$\alpha_p = 1 - \frac{\tilde{u}'_{i,\mathrm{P}}}{u'_{i,\mathrm{P}}} . \tag{7.48}$$

We can calculate $\tilde{u}'_{i,\mathrm{P}}$ using Eq. (7.38) but, in multi-dimensional problems, we would have more than one equation from which α_p can be calculated. However, if instead of calculating $\tilde{u}'_{i,\mathrm{P}}$, we use the approximation (7.41) used in SIMPLEC, then the above equation reduces to:

---

[1] Raithby and Schneider (1979) found this relation following a different route.

$$\alpha_p = 1 + \frac{\sum_l A_l^{u_i}}{A_{\mathrm{P}}^{u_i}} \ . \tag{7.49}$$

We may now recall that all conservative schemes, in the absence of any contribution to A_{P} from source terms, lead to $A_{\mathrm{P}} = -\sum_l A_l + A_{\mathrm{P}}^{\mathrm{t}}$, where $A_{\mathrm{P}}^{\mathrm{t}}$ is the contribution from the unsteady term. If a steady solution is sought through iterating for an infinite time step, $A_{\mathrm{P}}^{\mathrm{t}} = 0$, but we have to use under-relaxation, as explained in Sect. 5.4.2. In that case, $A_{\mathrm{P}} = -\sum_l A_l/\alpha_u$, where α_u is the under-relaxation factor for velocities (usually the same for all components, but it need not be so). We then obtain:

$$\alpha_p = 1 - \alpha_u \ , \tag{7.50}$$

which has been found to be nearly optimum and yields almost the same convergence rate for outer iterations as the SIMPLEC method.

The solution algorithm for this class of methods can be summarized as follows:

1. Start calculation of the fields at the new time t_{n+1} using the latest solution u_i^n and p^n as starting estimates for u_i^{n+1} and p^{n+1}.
2. Assemble and solve the linearized algebraic equation systems for the velocity components (momentum equations) to obtain u_i^{m*}.
3. Assemble and solve the pressure-correction equation to obtain p'.
4. Correct the velocities and pressure to obtain the velocity field u_i^m, which satisfies the continuity equation, and the new pressure p^m.
   For the PISO algorithm, solve the second pressure-correction equation and correct both velocities and pressure again.
   For SIMPLER, solve the pressure equation for p^m after u_i^m is obtained above.
5. Return to step 2 and repeat, using u_i^m and p^m as improved estimates for u_i^{n+1} and p^{n+1}, until all corrections are negligibly small.
6. Advance to the next time step.

Methods of this kind are fairly efficient for solving steady state problems; their convergence can be improved by the multigrid strategy, as will be demonstrated in Chap. 11. There are many derivatives of the above methods which are named differently, but they all have roots in the ideas described above and will not be listed here. We shall show below that the artificial compressibility method can also be interpreted in a similar way.

7.4 Other Methods

7.4.1 Fractional Step Methods

In the methods of the preceding section, the pressure is used to enforce continuity. It is also used in computing the velocity field in the first step of the

method; in this step, the pressure is treated explicitly. Why use it at all? The fractional step method of Kim and Moin (1985) provides an approach that does not use pressure in the predictor step. It is also important to recall that the role of the pressure in an incompressible flow is to enforce continuity; in some sense, it is more a mathematical variable than a physical one.

The fractional step concept is more a generic approach than a particular method. It is based on ideas similar to those that led to the alternating direction implicit method in Chap. 5. It is essentially an approximate factorization of a method; the underlying method need not be implicit. To see how this might work, we take the simplest case, the Euler explicit advancement of the Navier-Stokes equations in symbolic form:

$$u_i^{n+1} = u_i^n + (C_i + D_i + P_i)\Delta t \tag{7.51}$$

where C_i, D_i, and P_i represent the convective, diffusive and pressure terms, respectively. This equation is readily split into a three step method:

$$u_i^* = u_i^n + (C_i)\Delta t \tag{7.52}$$

$$u_i^{**} = u_i^* + (D_i)\Delta t \tag{7.53}$$

$$u_i^{n+1} = u_i^{**} + (P_i)\Delta t \tag{7.54}$$

In the third step, P_i is the gradient of a quantity that obeys a Poisson equation; naturally, this quantity must be chosen so that the continuity equation is satisfied. Depending on the particulars of the method, the source term in this Poisson equation may differ slightly from the source term in the standard Poisson equation for the pressure (7.21); for this reason, the variable is called the pseudo-pressure or a pressure-like variable. Also, note that it is possible to split the convective and diffusive terms further; for example, they may be split into their components in the various coordinate directions. Clearly, many basic methods can be used and many kinds of splitting can be applied to each.

We now present a particular fractional step method; again, many variations are possible.

For unsteady flows, a time accurate method such as a third or fourth order Runge-Kutta method (if an explicit method suffices) or the Crank-Nicolson or second order backward method (if more stability is required) is used. For steady flows, in order to take a large time step, an implicit method should be used; linearization and an ADI method may be used to solve the equations. Spatial discretization can be of any type described above. We shall use the semi-discrete form of equations and the Crank-Nicolson scheme; a similar method based on central difference approximations in space was used by Choi and Moin (1994) for direct simulations of turbulence.

In the first step, the velocity is advanced using pressure from the previous time step; convective terms, viscous terms, and body forces (if present) are represented by an equal blend of old and new values (Crank-Nicolson method in this particular case):

$$\frac{(\rho u_i)^* - (\rho u_i)^n}{\Delta t} = \frac{1}{2}\left[H(u_i^n) + H(u_i^*)\right] - \frac{\delta p^n}{\delta x_i} , \tag{7.55}$$

where $H(u_i)$ is an operator representing the discretized convective, diffusive, and source terms. This system of equations must be solved for u_i^*; any method can be used. Unless the time step is very small, one should iterate to account for the non-linearity of the equations; Choi et al. (1994) used a Newton iterative method.

In the second step, half of the old pressure gradient is removed from u_i^*, leading to u_i^{**}:

$$\frac{(\rho u_i)^{**} - (\rho u_i)^*}{\Delta t} = \frac{1}{2}\frac{\delta p^n}{\delta x_i} . \tag{7.56}$$

The final velocity at the new time level requires the gradient of the (as yet unknown) new pressure:

$$\frac{(\rho u_i)^{n+1} - (\rho u_i)^{**}}{\Delta t} = -\frac{1}{2}\frac{\delta p^{n+1}}{\delta x_i} . \tag{7.57}$$

The requirement that the new velocity satisfy the continuity equation leads to a Poisson equation for the new pressure:

$$\frac{\delta}{\delta x_i}\left(\frac{\delta p^{n+1}}{\delta x_i}\right) = \frac{2}{\Delta t}\frac{\delta(\rho u_i)^{**}}{\delta x_i} . \tag{7.58}$$

Upon solution of the pressure equation, the new velocity field is obtained from Eq. (7.57). It satisfies the continuity equation and the momentum equation in the form:

$$\frac{(\rho u_i)^{n+1} - (\rho u_i)^n}{\Delta t} = \frac{1}{2}\left[H(u_i^n) + H(u_i^*)\right] - \frac{1}{2}\left(\frac{\delta p^n}{\delta x_i} + \frac{\delta p^{n+1}}{\delta x_i}\right) . \tag{7.59}$$

For this equation to represent the Crank–Nicolson method correctly, $H(u_i^*)$ should be replaced by $H(u_i^{n+1})$. However, from Eqs. (7.56) and (7.57) one can easily show that the error is of second order in time and thus consistent with other errors:

$$u_i^{n+1} - u_i^* = -\frac{\Delta t}{2\rho}\frac{\delta(p^{n+1} - p^n)}{\delta x_i} \approx \frac{(\Delta t)^2}{2\rho}\frac{\delta}{\delta x_i}\left(\frac{\delta p}{\delta t}\right) . \tag{7.60}$$

Note that, by subtracting Eq. (7.55) from Eq. (7.59), one obtains an equation for the pressure correction $p' = p^{n+1} - p^n$:

$$\frac{(\rho u_i)^{n+1} - (\rho u_i)^*}{\Delta t} = -\frac{1}{2}\frac{\delta p'}{\delta x_i} . \tag{7.61}$$

The Poisson equation for p' has the same form as Eq. (7.58), except that u_i^{**} is replaced by u_i^*.

Fractional step methods have become rather popular. There is a wide variety of them, due to a vast choice of approaches to time and space discretization; however, they are all based on the principles described above.

The major difference between the fractional-step method and pressure-correction methods of the SIMPLE-type is that in the former, the pressure (or pressure-correction) equation is solved once per time step, while in the latter, both the momentum and pressure-correction equations are solved several times within each time step (outer iterations). This is largely because fractional step methods are used mainly in unsteady flow simulations while the latter are used predominantly to compute steady flows. Since, in SIMPLE-type methods, mass conservation is enforced only at the end of a time step, the pressure-correction equation need not be solved accurately on each outer iteration (reduction of the residual by one order of magnitude usually suffices). Indeed, for steady flows, accurate satisfaction of the continuity condition is required only at convergence. In simulations of unsteady flows, the pressure (or pressure-correction) equation must be solved to a tight tolerance to ensure mass conservation at each time step. Multigrid or spectral methods are usually used to solve the Poisson equation for the pressure in unsteady flow simulations in simple geometries while, for steady flows or complex geometries, the linear equations are usually solved using conjugate-gradient methods.

If the time step is large, the fractional step method produces an error due to the operator splitting, as shown in Eq. (7.60). This error can be eliminated either by reducing the time step or by using iteration of the kind used in SIMPLE-type methods. However, if the splitting error is significant, the temporal discretization error is also large. Therefore, reducing the time step is the most appropriate means of improving accuracy. Note that the PISO-method introduced in the preceding section is very similar to the fractional-step method and has a splitting error proportional to $(\Delta t)^2$.

7.4.2 Streamfunction-Vorticity Methods

For incompressible two-dimensional flows with constant fluid properties, the Navier-Stokes equations can be simplified by introducing the *streamfunction* ψ and *vorticity* ω as dependent variables. These two quantities are defined in terms of Cartesian velocity components by:

$$\frac{\partial \psi}{\partial y} = u_x \;, \qquad \frac{\partial \psi}{\partial x} = -u_y \;, \tag{7.62}$$

and

$$\omega = \frac{\partial u_y}{\partial x} - \frac{\partial u_x}{\partial y} \;. \tag{7.63}$$

Lines of constant ψ are streamlines (lines which are everywhere parallel to the flow), giving this variable its name. The vorticity is associated with rotational

motion; Eq. (7.63) is a special case of the more general definition that applies in 3D as well:

$$\boldsymbol{\omega} = \nabla \times \boldsymbol{v} \ . \tag{7.64}$$

In two dimensional flows, the vorticity vector is orthogonal to the plane of flow and Eq. (7.64) reduces to Eq. (7.63). The principal reason for introducing the streamfunction is that, for flows in which ρ, μ and $\boldsymbol{g}$ are constant, the continuity equation is identically satisfied and need not be dealt with explicitly. Substitution of Eqs. (7.62) into the definition of the vorticity (7.63) leads to a kinematic equation connecting the streamfunction and the vorticity:

$$\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = -\omega \ . \tag{7.65}$$

Finally, by differentiating the x and y momentum equations with respect to y and x, respectively, and subtracting the results from each other we obtain the dynamic equation for the vorticity:

$$\rho \frac{\partial \omega}{\partial t} + \rho u_x \frac{\partial \omega}{\partial x} + \rho u_y \frac{\partial \omega}{\partial y} = \mu \left(\frac{\partial^2 \omega}{\partial x^2} + \frac{\partial^2 \omega}{\partial y^2} \right) \ . \tag{7.66}$$

The pressure does not appear in either of these equations i.e. it has been eliminated as a dependent variable. Thus the Navier-Stokes equations have been replaced by a set of just two partial differential equations, in place of the three for the velocity components and pressure. This reduction in the number of dependent variables and equations is what makes this approach attractive.

The two equations are coupled through the appearance of u_x and u_y (which are derivatives of ψ) in the vorticity equation and by the vorticity ω acting as the source term in the Poisson equation for ψ. The velocity components are obtained by differentiating the streamfunction. If it is needed, the pressure can be obtained by solving the Poisson equation as described in Sect. 7.3.1.

A solution method for these equations is the following. Given an initial velocity field, the vorticity is computed by differentiation. The dynamic vorticity equation is then used to compute the vorticity at the new time step; any standard time advance method may be used for this purpose. Having the vorticity, it is possible to compute the streamfunction at the new time step by solving the Poisson equation; any iterative scheme for elliptic equations may be used. Finally, having the streamfunction, the velocity components are easily obtained by differentiation and we are ready to begin the calculation for the next time step.

A problem with this approach lies in the boundary conditions, especially in complex geometries. Since the flow is parallel to them, solid boundaries and symmetry planes are surfaces of constant streamfunction. However, the values of the streamfunction at these boundaries can be calculated only if

velocities are known. A more difficult problem is that neither vorticity nor its derivatives at the boundary are usually known in advance. For example, vorticity at a wall is equal to $\omega_{\text{wall}} = -\tau_{\text{wall}}/\mu$, where τ_{wall} is the wall shear stress, which is usually the quantity we seek to determine. Boundary values of the vorticity may be calculated from the streamfunction by differentiation using one-sided finite differences in the direction normal to the boundary, see Eq. (7.65). This approach usually slows down the convergence rate. The vorticity is singular at sharp corners of the boundary and special care is required in treating them. For example, at the corners labeled A and B in Fig. 7.3 the derivatives of $\partial u_y/\partial x$ and $\partial u_x/\partial y$ are not continuous, which means that the vorticity ω is also not continuous there and cannot be computed using the approach described above. Some authors extrapolate the vorticity from the interior to the boundary but this does not provide a unique result at A and B either. It is possible to derive analytical behavior of the vorticity near a corner and use it to correct the solution but this is difficult as each special case must be treated separately. A simpler but efficient way of avoiding large errors (which may be convected downstream) is to locally refine the grid around singularities.
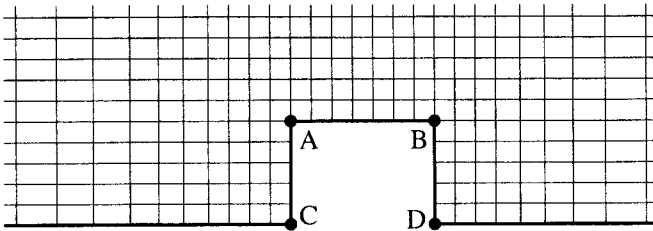
Fig. 7.3. Finite difference grid for the calculation of flow over a rib, showing protruding corners A and B where special treatment is necessary to determine boundary values of vorticity

The vorticity-streamfunction approach has seen considerable use for two-dimensional incompressible flows. It has become less popular in recent years because its extension to three-dimensional flows is difficult. Both the vorticity and streamfunction become three-component vectors in three dimensions so one has a system of six partial differential equations in place of the four that are necessary in a velocity-pressure formulation. It also inherits the difficulties in dealing with variable fluid properties, compressibility, and boundary conditions that were described above for two dimensional flows.

7.4.3 Artificial Compressibility Methods

Compressible flow is an area of fluid mechanics of great importance. Its applications, especially in aerodynamics and turbine engine design, have caused

a great deal of attention to be focused on development of methods for the numerical solution of the compressible flow equations. Many such methods have been developed. An obvious question is whether they can be adapted to the solution of the incompressible flows. Clearly, we cannot cover the full range of methods for compressible flow in this work; instead we shall focus on how these methods may be adapted to incompressible flow and a description of a few of the key properties of artificial compressibility methods.

The major difference between the equations of compressible flow and those of incompressible flow is their mathematical character. The compressible flow equations are hyperbolic which means that they have real characteristics on which signals travel at finite propagation speeds; this reflects the ability of compressible fluids to support sound waves. By contrast, we have seen that the incompressible equations have a mixed parabolic-elliptic character. If methods for compressible flow are to be used to compute incompressible flow, the character of the equations will need to be modified.

The difference in character can be traced to the lack of a time derivative term in the incompressible continuity equation. The compressible version contains the time derivative of the density. So, the most straight-forward means of giving the incompressible equations hyperbolic character is to append a time derivative to the continuity equation. Since density is constant, adding $\partial\rho/\partial t$, i.e. using the compressible equation, is not possible. Time derivatives of velocity components appear in the momentum equations so they are not logical choices. That leaves the time derivative of the pressure as the clear choice.

Addition of a time derivative of the pressure to the continuity equation means that we are no longer solving the true incompressible equations. As a result, the time history generated cannot be accurate and the applicability of artificial compressibility methods to unsteady incompressible flows is a questionable enterprise although it has been attempted. On the other hand, at convergence, the time derivative is zero and the solution satisfies the incompressible equations. This approach was first proposed by Chorin (1967) and a number of versions differing mainly in the underlying compressible flow method used, have been presented in the literature. As noted above, the essential idea is to add a time derivative of pressure to the continuity equation:

$$\frac{1}{\beta}\frac{\partial p}{\partial t} + \frac{\partial(\rho u_i)}{\partial x_i} = 0 \,, \tag{7.67}$$

where β is an artificial compressibility parameter whose value is key to the performance of this method. Clearly, the larger the value of β, the more "incompressible" the equations; however, large β makes the equations very stiff numerically. We shall consider only the case of constant density, but the method can be applied to variable density flows.

To connect this method with the ones described above, we note that the intermediate velocity field $(u_i^*)^{n+1}$, obtained from the momentum equations using old pressure, does not satisfy the incompressible continuity equation, i.e.,

$$\left[\frac{\delta(\rho u_i^*)^{n+1}}{\delta x_i}\right]_{\mathrm{P}} = \Delta\dot{m}_{\mathrm{P}} , \tag{7.68}$$

at the time step $n+1$. The derivatives on the left hand side of this equation are evaluated by some finite difference approximation; the choice is not important here, which is why symbolic difference notation is used.

For solving these equations, many methods are available. In fact, because each equation now contains a time derivative, methods employed to solve them can be modeled after ones used to solve ordinary differential equations presented in Chap. 6. Because the artificial compressibility method is principally intended for steady flows, implicit methods should be favored. Another important point is that the principal difficulty faced in compressible flow, namely the possibility of transition from subsonic to supersonic flow and, especially, the possible existence of shock waves can be avoided. The best choice for a solution method for two or three dimensional problems is an implicit method that does not require the solution of a full two or three dimensional problem at each time step, which means that an alternating direction implicit or approximate factorization method is the best choice. An example of a scheme for deriving a pressure equation using artificial compressibility is presented below.

The simplest scheme uses a first order explicit discretization in time; it enables pointwise calculation of pressure, but places a severe restriction on the size of time step. Since the time development of pressure is not important and we are interested in obtaining the steady state solution as quickly as possible, the fully implicit Euler scheme is a better choice:

$$\frac{p_{\mathrm{P}}^{n+1} - p_{\mathrm{P}}^{n}}{\beta\,\Delta t} + \left[\frac{\delta(\rho u_i)}{\delta x_i}\right]_{\mathrm{P}}^{n+1} = 0 . \tag{7.69}$$

The problem is that the velocity field at the new time level is not known. However, one can linearize about the known old state and transform the above equation into a Poisson equation for pressure or pressure correction! Let us see how this can be done.

The unknown quantity $(\rho u_i)^{n+1}$ may be approximated as:

$$(\rho u_i)^{n+1} \approx (\rho u_i^*)^{n+1} + \left[\frac{\partial(\rho u_i^*)}{\partial p}\right]^{n+1} (p^{n+1} - p^n) . \tag{7.70}$$

By inserting this expression into the continuity equation (7.69) we obtain an equation for the new pressure p^{n+1}.

Let us now see how the artificial compressibility approach may be used to derive a pressure-correction equation similar to those described in preceding sections.

We may recall from the previous section that from the implicitly discretized momentum equations, one can demonstrate the following relationship between ρu_i and the *pressure gradient* (see Eq. (7.32)):

$$(\rho u_i^*)_{\mathrm{P}}^{n+1} = (\rho \tilde{u}_i^*)_{\mathrm{P}}^{n+1} - \frac{\rho}{A_{\mathrm{P}}^{u_i}} \left(\frac{\delta p^n}{\delta x_i} \right)_{\mathrm{P}} . \tag{7.71}$$

Instead of Eq. (7.70) we postulate the following expression for $(\rho u_i)^{n+1}$:

$$(\rho u_i)^{n+1} \approx (\rho u_i^*)^{n+1} + \left[\frac{\partial (\rho u_i^*)}{\partial \left(\frac{\delta p}{\delta x_i} \right)} \right]^{n+1} \left(\frac{\delta p^{n+1}}{\delta x_i} - \frac{\delta p^n}{\delta x_i} \right) . \tag{7.72}$$

From Eq. (7.71) we find that:

$$d_{\mathrm{P}}^i = \left[\frac{\partial (\rho u_i^*)}{\partial \left(\frac{\delta p}{\delta x_i} \right)} \right]_{\mathrm{P}}^{n+1} = - \frac{\rho}{A_{\mathrm{P}}^{u_i}} . \tag{7.73}$$

If we further introduce the pressure correction

$$p' = p^{n+1} - p^n , \tag{7.74}$$

we may finally rewrite the Eq. (7.72) as:

$$(\rho u_i)^{n+1} = (\rho u_i^*)^{n+1} + d_{\mathrm{P}}^i \frac{\delta p'}{\delta x_i} . \tag{7.75}$$

Obviously, if the densities in $(\rho u_i)^{n+1}$ and $(\rho u_i^*)^{n+1}$ are the same, this implies a velocity correction of the same form as in the SIMPLE method, see Eq. (7.37), i.e.:

$$u_i' = - \frac{1}{A_{\mathrm{P}}^{u_i}} \frac{\delta p'}{\delta x_i} . \tag{7.76}$$

By substituting expression (7.75) into Eq. (7.69), we obtain:

$$\frac{p_{\mathrm{P}}'}{\beta \Delta t} + \left[\frac{\delta (\rho u_i^*)^{n+1}}{\delta x_i} + \frac{\delta}{\delta x_i} \left(d^i \frac{\delta p'}{\delta x_i} \right) \right]_{\mathrm{P}} = 0 , \tag{7.77}$$

which can, with the help of Eq. (7.68), be further rearranged to give:

$$\frac{p'_{\mathrm{P}}}{\beta \Delta t} + \frac{\delta}{\delta x_i}\left(d^i \frac{\delta p'}{\delta x_i}\right)_{\mathrm{P}} = -\Delta \dot{m}_{\mathrm{P}} \, . \tag{7.78}$$

This equation is very similar to the pressure-correction equation of the SIMPLE method, Eq. (7.39). It thus appears that all the pressure calculation methods presented so far, although arrived at via different routes, reduce to the same basic method. Again, it is important that the pressure derivatives inside brackets are approximated in the same way as in the momentum equations, while the outer derivative is the one from the continuity equation.

The crucial factor for convergence of a method based on artificial compressibility is the choice of parameter β. The optimum value is problem dependent, although some authors have suggested an automatic procedure for choosing it. A very large value would require the corrected velocity field to satisfy the incompressible continuity equation. In the above version of the method, this corresponds to the SIMPLE scheme without under-relaxation of pressure correction; the procedure would then converge only for small Δt. However, if only a portion of p' is added to the pressure as in SIMPLE, infinitely large β could be used.

On the other hand, the lowest value of β allowed can be determined by looking at the propagation speed of pressure waves. The pseudo-sound speed is:

$$c = \sqrt{v^2 + \beta} \, .$$

By requiring pressure waves to propagate much faster than the vorticity spreads, the following criterion can be derived for a simple channel flow (see Kwak et al., 1986):

$$\beta \gg \left[1 + \frac{4}{\mathrm{Re}}\left(\frac{x_{\mathrm{ref}}}{x_\delta}\right)^2 \left(\frac{x_L}{x_{\mathrm{ref}}}\right)\right]^2 - 1 \, ,$$

where x_L is the distance between inlet and outlet, x_δ is half the distance between two walls and x_{ref} is the reference length. Typical values of β used in various methods based on artificial compressibility were in the range between 0.1 and 10.

Obviously, $1/(\beta \Delta t)$ should be small compared to the coefficients arising from the second term in Eq. (7.78) if the corrected velocity field is to closely satisfy the continuity equation. This is a necessity if rapid convergence is to be obtained. For some iterative solution methods (e.g. using domain decomposition technique in parallel processing or block-structured grids in complex geometries) it has been found useful to divide the A_{P} coefficient of the pressure-correction equation in SIMPLE by a factor smaller than unity (0.95 to 0.99). This is equivalent to the artificial compressibility method with $1/(\beta \Delta t) \approx (0.01 \text{ to } 0.05) A_{\mathrm{P}}$.

7.5 Solution Methods for the Navier-Stokes Equations

We have described discretization methods for the various terms in the transport equations. The linkage of the pressure and the velocity components in incompressible flows was demonstrated and a few solution methods have been given. Many other methods of solving the Navier-Stokes equations can be devised. It is impossible to describe all of them here. However, many of them have elements in common with the methods already described. Familiarity with these methods should allow the reader to understand the others.

We describe below in some detail two methods that are representative of a larger group of methods. First an implicit method using the pressure-correction equation and staggered grids is described in enough detail to allow straightforward translation into a computer code. The corresponding code is available via Internet; see Appendix for details.

7.5.1 Implicit Scheme Using Pressure-Correction and a Staggered Grid

In this section we present an implicit finite volume scheme that uses the pressure-correction method on a staggered two-dimensional Cartesian grid. Solution in complicated geometries is described in the next chapter.

The Navier-Stokes equations in integral form read:

$$\int_S \rho \boldsymbol{v} \cdot \boldsymbol{n} \, \mathrm{d}S = 0 \, , \tag{7.79}$$

$$\frac{\partial}{\partial t} \int_\Omega \rho u_i \, \mathrm{d}\Omega + \int_S \rho u_i \boldsymbol{v} \cdot \boldsymbol{n} \, \mathrm{d}S = \int_S \tau_{ij} \boldsymbol{i}_j \cdot \boldsymbol{n} \, \mathrm{d}S - \int_S p \boldsymbol{i}_i \cdot \boldsymbol{n} \, \mathrm{d}S + \int_\Omega (\rho - \rho_0) g_i \, \mathrm{d}\Omega \, . \tag{7.80}$$

For convenience, it is assumed that the only body force is buoyancy. The macroscopic momentum flux vector $\boldsymbol{t}_i$, see Eq. 1.18, is split into a viscous contribution $\tau_{ij}\boldsymbol{i}_j$ and a pressure contribution $p\boldsymbol{i}_i$. We assume the density constant except in the buoyancy term, i.e. we use the Boussinesq approximation. The mean gravitational force is incorporated into the pressure term, as shown in Sect. 1.4.

Typical staggered control volumes are shown in Fig. 7.4. The control volumes for u_x and u_y are displaced with respect to the control volume for the continuity equation. For non-uniform grids, the velocity nodes are not at the centers of their control volumes. Cell faces 'e' and 'w' for u_x and 'n' and 's' for u_y lie midway between the nodes. For convenience, we shall sometimes use u instead of u_x and v instead of u_y.
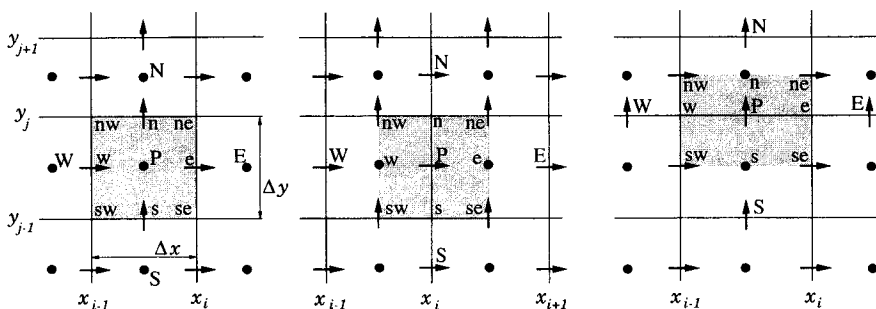
Fig. 7.4. Control volumes for a staggered grid: for mass conservation and scalar quantities (left), for x-momentum (center) and for y-momentum (right)

We will use the second order implicit three time level scheme described in Sect. 6.2.4 for integration in time. This leads to the following approximation of the unsteady term:

$$\left[\frac{\partial}{\partial t}\int_{\Omega} \rho u_i \,\mathrm{d}\Omega\right]_{\mathrm{P}} \approx \frac{\rho\,\Delta\Omega}{2\,\Delta t}\left(3u_i^{n+1} - 4u_i^n + u_i^{n-1}\right) =$$

$$A_{\mathrm{P}}^{\mathrm{t}} u_{i,\mathrm{P}}^{n+1} - Q_{u_i}^{\mathrm{t}} \;, \tag{7.81}$$

where

$$A_{\mathrm{P}}^{\mathrm{t}} = \frac{3\rho\,\Delta\Omega}{2\Delta t} \quad \text{and} \quad Q_{u_i}^{\mathrm{t}} = \frac{\rho\,\Delta\Omega}{2\Delta t}\left(4u_i^n - u_i^{n-1}\right) \;. \tag{7.82}$$

We shall drop the superscript $n+1$; all terms are evaluated at t_{n+1} unless stated otherwise. Because the scheme is implicit, the equations require iterative solution. If the time steps are as small as those used in explicit schemes, one or two iterations per time step will suffice. For flows with slow transients, we may use larger time steps and more iterations will be necessary. As noted earlier, these iterations are called outer iterations to distinguish them from the inner iterations used to solve linear equations such as the pressure correction equation. We assume that one of the solvers described in Chap. 5 is used for the latter and shall concentrate on the outer iterations.

We now consider the approximation of the convective and diffusive fluxes and the source terms. The surface integrals may be split into four CV face integrals. Let us concentrate attention on CV face 'e'; the other faces are treated in the same way, and the results can be obtained by index substitution. We shall adopt the second order central difference approximations presented in Chap. 4. Fluxes are approximated by assuming that the value of a quantity at a CV face center represents the mean value over the face (midpoint rule approximation). On the mth outer iteration, all nonlinear terms are approximated by a product of an 'old' (from the preceding outer iteration) and a 'new' value. Thus, in discretizing the momentum equations, the

mass flux through each CV face is evaluated using the existing velocity field and is assumed known:

$$\dot{m}_{\mathrm{e}}^{m} = \int_{S_{\mathrm{e}}} \rho \boldsymbol{v} \cdot \boldsymbol{n} \, \mathrm{d}S \approx (\rho u)_{\mathrm{e}}^{m-1} S_{\mathrm{e}} \,. \tag{7.83}$$

This kind of linearization is essentially the first step of Picard iteration; other linearizations for implicit schemes were described in Chap. 5. Unless specifically stated otherwise, all variables in the remainder of this section belong to the mth outer iteration. The mass fluxes (7.83) satisfy the continuity equation on the 'scalar' CV, see Fig. 7.4. Mass fluxes at the faces of the momentum CVs must be obtained by interpolation; ideally, these fluxes would provide mass conservation for the momentum CV but this can be guaranteed only to the accuracy of the interpolation. Another possibility is to use the mass fluxes from the scalar CV faces. Since the east and west faces of a u-CV are halfway between scalar CV faces, the mass fluxes can be calculated as:

$$\dot{m}_{\mathrm{e}}^{u} = \frac{1}{2}(\dot{m}_{\mathrm{P}} + \dot{m}_{\mathrm{E}})^{u} \; ; \quad \dot{m}_{\mathrm{w}}^{u} = \frac{1}{2}(\dot{m}_{\mathrm{W}} + \dot{m}_{\mathrm{P}})^{u} \,. \tag{7.84}$$

The mass fluxes through the north and south faces of the u-CV can be approximated as half the sum of the two scalar CV face mass fluxes:

$$\dot{m}_{\mathrm{n}}^{u} = \frac{1}{2}(\dot{m}_{\mathrm{ne}} + \dot{m}_{\mathrm{nw}})^{u} \; ; \quad \dot{m}_{\mathrm{s}}^{u} = \frac{1}{2}(\dot{m}_{\mathrm{se}} + \dot{m}_{\mathrm{sw}})^{u} \,. \tag{7.85}$$

The superscript u denotes that the indices refer to the u-CV, see Fig. 7.4. The sum of the four mass fluxes for the u-CV is thus half the sum of mass fluxes into the two adjacent scalar CVs. They therefore satisfy the continuity equation for the double scalar CV so the mass fluxes through the u-CV faces also conserve mass. This result also holds for v-momentum CVs. It is necessary to ensure that the mass fluxes through the momentum CVs satisfy the continuity equation; otherwise, momentum will not be conserved.

The convective flux of u_i-momentum through the 'e'-face of a u-CV is then (see Sect. 4.2 and Eq. (7.83)):

$$F_{i,\mathrm{e}}^{\mathrm{c}} = \int_{\mathrm{S}} \rho u_i \boldsymbol{v} \cdot \boldsymbol{n} \, \mathrm{d}S \approx \dot{m}_{\mathrm{e}} u_{i,\mathrm{e}} \,. \tag{7.86}$$

The CV face value of u_i used in this expression need not be the one used to calculate the mass flux, although an approximation of the same accuracy is desirable. Linear interpolation is the simplest second order approximation. We call this a *central difference scheme* (CDS), although no differencing is involved. This is because, on uniform grids, it results in the same algebraic equations as the CDS finite difference method.

Some iterative solvers fail to converge when applied to the algebraic equation systems derived from central difference approximations of convective

fluxes. This is because the matrices may not be diagonally dominant; these equations are best solved using *deferred correction* approach described in Sect. 5.6. In this method, the flux is expressed as:

$$F^{\mathrm{c}}_{i,\mathrm{e}} = \dot{m}_{\mathrm{e}} u^{\mathrm{UDS}}_{i,\mathrm{e}} + \dot{m}_{\mathrm{e}} (u^{\mathrm{CDS}}_{i,e} - u^{\mathrm{UDS}}_{i,e})^{m-1} , \tag{7.87}$$

where superscripts CDS and UDS denote approximation by central and upwind differences, respectively (see Sect. 4.4). The term in brackets is evaluated using values from the previous iteration while the matrix is computed using the UDS-approximation. At convergence, the UDS contributions cancel out, leaving a CDS solution. This procedure usually converges at approximately the rate obtained for a pure upwind approximation.

The two schemes may also be blended; this is achieved by multiplying the explicit part (the term in brackets in Eq. (7.87)) by a factor $0 \leq \beta \leq 1$. This practice can remove the oscillations obtained with central differences on coarse grids. However, it improves the esthetics of the results at the cost of decreasing the accuracy. Blending may be used locally, e.g. to allow calculation of flows with shocks with CDS; this is preferable to applying it everywhere.

Calculation of the diffusive fluxes requires evaluation of the stresses τ_{xx} and τ_{yx} at the CV face 'e'. Since the outward unit normal vector at this CV face is $\boldsymbol{i}$, we have:

$$F^{\mathrm{d}}_{i,\mathrm{e}} = \int_{S_\mathrm{e}} \tau_{ix} \, \mathrm{d}S \approx (\tau_{ix})_\mathrm{e} S_\mathrm{e} , \tag{7.88}$$

where $S_\mathrm{e} = y_j - y_{j-1} = \Delta y$ for the u-CV and $S_\mathrm{e} = \frac{1}{2}(y_{j+1} - y_{j-1})$ for the v-CV. The stresses at CV face require approximation of the derivatives; central difference approximations lead to:

$$(\tau_{xx})_\mathrm{e} = 2 \left(\mu \frac{\partial u}{\partial x} \right)_\mathrm{e} \approx 2\mu \frac{u_\mathrm{E} - u_\mathrm{P}}{x_\mathrm{E} - x_\mathrm{P}} , \tag{7.89}$$

$$(\tau_{yx})_\mathrm{e} = \mu \left(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y} \right)_\mathrm{e} \approx \mu \frac{v_\mathrm{E} - v_\mathrm{P}}{x_\mathrm{E} - x_\mathrm{P}} + \mu \frac{u_\mathrm{ne} - u_\mathrm{se}}{y_\mathrm{ne} - y_\mathrm{se}} . \tag{7.90}$$

Note that τ_{xx} is evaluated at the 'e' face of the u-CV, and τ_{yx} at the 'e' face of the v-CV, so the indices refer to locations on the appropriate CVs, see Fig. 7.4. Thus, u_ne and u_se on the v-CV are actually nodal values of the u-velocity and no interpolation is necessary.

At the other CV faces we obtain similar expressions. For u-CVs, we need to approximate τ_{xx} at the faces 'e' and 'w', and τ_{xy} at the faces 'n' and 's'. For v-CVs, τ_{yx} is needed at the faces 'e' and 'w' and τ_{yy} at the faces 'n' and 's'.

The pressure terms are approximated by:

$$Q^{\mathrm{p}}_u = -\int_S p \boldsymbol{i} \cdot \boldsymbol{n} \, \mathrm{d}S \approx -(p_\mathrm{e} S_\mathrm{e} - p_\mathrm{w} S_\mathrm{w})^{m-1} \tag{7.91}$$

for the u-equation and

$$Q_v^{\mathrm{p}} = -\int_S p\boldsymbol{j} \cdot \boldsymbol{n}\, \mathrm{d}S \approx -(p_{\mathrm{n}}S_{\mathrm{n}} - p_{\mathrm{s}}S_{\mathrm{s}})^{m-1} \tag{7.92}$$

for the v-equation. There are no pressure force contributions from the 'n' and 's' CV faces to the u-equation or from the 'e' and 'w' faces to the v-equation on a Cartesian grid.

If buoyancy forces are present, they are approximated by:

$$Q_{u_i}^{\mathrm{b}} = \int_\Omega (\rho - \rho_0) g_i\, \mathrm{d}\Omega \approx (\rho_{\mathrm{P}}^{m-1} - \rho_0) g_i\, \Delta\Omega \;, \tag{7.93}$$

where $\Delta\Omega = (x_{\mathrm{e}} - x_{\mathrm{w}})(y_{\mathrm{n}} - y_{\mathrm{s}}) = \frac{1}{2}(x_{i+1} - x_{i-1})(y_j - y_{j-1})$ for the u-CV and $\Delta\Omega = \frac{1}{2}(x_i - x_{i-1})(y_{j+1} - y_{j-1})$ for the v-CV. Any other body force can be approximated in the same way.

The approximation to the complete u_i-momentum equation is:

$$A_{\mathrm{P}}^{\mathrm{t}} u_{i,\mathrm{P}} + F_i^{\mathrm{c}} = F_i^{\mathrm{d}} + Q_i^{\mathrm{p}} + Q_i^{\mathrm{b}} + Q_i^{\mathrm{t}} \;, \tag{7.94}$$

where

$$F^{\mathrm{c}} = F_{\mathrm{e}}^{\mathrm{c}} + F_{\mathrm{w}}^{\mathrm{c}} + F_{\mathrm{n}}^{\mathrm{c}} + F_{\mathrm{s}}^{\mathrm{c}} \quad \text{and} \quad F^{\mathrm{d}} = F_{\mathrm{e}}^{\mathrm{d}} + F_{\mathrm{w}}^{\mathrm{d}} + F_{\mathrm{n}}^{\mathrm{d}} + F_{\mathrm{s}}^{\mathrm{d}} \;. \tag{7.95}$$

If ρ and μ are constant, part of the diffusive flux term cancels out by virtue of the continuity equation, see Sect. 7.1. (It may not exactly cancel out in the numerical approximation but the equations may be simplified by deleting those terms prior to discretization). For example, in the u-equation, the τ_{xx} term on the 'e' and 'w' faces will be reduced by half, and in the τ_{yx} term at the 'n' and 's' faces, the $\partial v/\partial x$ contribution is removed. Even when ρ and μ are not constant, the sum of these terms contributes in only a minor way to F^{d}. This is why an explicit 'diffusive source term', e.g. for u:

$$Q_u^{\mathrm{d}} = \left[\mu_{\mathrm{e}} S_{\mathrm{e}} \frac{u_{\mathrm{E}} - u_{\mathrm{P}}}{x_{\mathrm{E}} - x_{\mathrm{P}}} - \mu_{\mathrm{w}} S_{\mathrm{w}} \frac{u_{\mathrm{P}} - u_{\mathrm{W}}}{x_{\mathrm{P}} - x_{\mathrm{W}}} + \right.$$
$$\left. \mu_{\mathrm{n}} S_{\mathrm{n}} \frac{v_{\mathrm{ne}} - v_{\mathrm{nw}}}{x_{\mathrm{ne}} - x_{\mathrm{nw}}} - \mu_{\mathrm{s}} S_{\mathrm{s}} \frac{v_{\mathrm{se}} - v_{\mathrm{sw}}}{x_{\mathrm{se}} - x_{\mathrm{sw}}} \right]^{m-1} \tag{7.96}$$

is usually calculated from the previous outer iteration $m-1$ and treated explicitly. Only $F^{\mathrm{d}} - Q_u^{\mathrm{d}}$ is treated implicitly. A consequence of this approximation is that, on a colocated grid, the matrix implied by Eq. (7.94) is identical for all three velocity components.

When the approximations for all the fluxes and source terms are substituted into Eq. (7.94), we obtain an algebraic equation of the form:

$$A_{\mathrm{P}}^{u} u_{\mathrm{P}} + \sum_l A_l^{u} u_l = Q_{\mathrm{P}}^{u} \;, \quad l = \mathrm{E}, \mathrm{W}, \mathrm{N}, \mathrm{S} \;. \tag{7.97}$$

The equation for v has the same form. The coefficients depend on the approximations used; for the CDS approximations applied above, the coefficients of the u equation are:

$$\begin{aligned}
A_{\mathrm{E}}^{u} &= \min(\dot{m}_{\mathrm{e}}^{u}, 0) - \frac{\mu_{\mathrm{e}} S_{\mathrm{e}}}{x_{\mathrm{E}} - x_{\mathrm{P}}} \,, \\
A_{\mathrm{N}}^{u} &= \min(\dot{m}_{\mathrm{n}}^{u}, 0) - \frac{\mu_{\mathrm{n}} S_{\mathrm{n}}}{y_{\mathrm{N}} - y_{\mathrm{P}}} \,, \\
A_{\mathrm{W}}^{u} &= \min(\dot{m}_{\mathrm{w}}^{u}, 0) - \frac{\mu_{\mathrm{w}} S_{\mathrm{w}}}{x_{\mathrm{P}} - x_{\mathrm{W}}} \,, \\
A_{\mathrm{S}}^{u} &= \min(\dot{m}_{\mathrm{s}}^{u}, 0) - \frac{\mu_{\mathrm{s}} S_{\mathrm{s}}}{y_{\mathrm{P}} - y_{\mathrm{S}}} \,, \\
A_{\mathrm{P}}^{u} &= A_{\mathrm{P}}^{\mathrm{t}} - \sum_{l} A_{l}^{u} \,, \quad l = \mathrm{E}, \mathrm{W}, \mathrm{N}, \mathrm{S} \,.
\end{aligned} \tag{7.98}$$

Note that $\dot{m}_{\mathrm{w}}$ for the CV centered around node P equals $-\dot{m}_{\mathrm{e}}$ for the CV centered around node W. The source term Q_{P}^{u} contains not only the pressure and buoyancy terms but also the portion of convective and diffusive fluxes resulting from deferred correction and the contribution of the unsteady term, i.e.:

$$Q_{\mathrm{P}}^{u} = Q_{u}^{\mathrm{p}} + Q_{u}^{\mathrm{b}} + Q_{u}^{\mathrm{c}} + Q_{u}^{\mathrm{d}} + Q_{u}^{\mathrm{t}} \,, \tag{7.99}$$

where

$$Q_{u}^{\mathrm{c}} = \left[(F_{u}^{\mathrm{c}})^{\mathrm{UDS}} - (F_{u}^{\mathrm{c}})^{\mathrm{CDS}} \right]^{m-1} . \tag{7.100}$$

This 'convective source' is calculated using the velocities from the previous outer iteration $m-1$.

The coefficients in the v equation are obtained in the same way and have the same form; however, the grid locations 'e', 'n' etc. have different coordinates, see Fig. 7.4.

The linearized momentum equations are solved with the sequential solution method (see Sect. 5.4), using the 'old' mass fluxes and the pressure from the previous outer iteration. This produces new velocities u^* and v^* which do not necessarily satisfy the continuity equation so:

$$\dot{m}_{\mathrm{e}}^{*} + \dot{m}_{\mathrm{w}}^{*} + \dot{m}_{\mathrm{n}}^{*} + \dot{m}_{\mathrm{s}}^{*} = \Delta \dot{m}_{\mathrm{P}}^{*} \,, \tag{7.101}$$

where the mass fluxes are calculated according to Eq. (7.83) using u^* and v^*. Since the arrangement of variables is staggered, the cell face velocities on a mass CV are the nodal values. The indices below refer to this CV, see Fig. 7.4, unless otherwise stated.

The velocity components u^* and v^* calculated from the momentum equations can be expressed as follows (by dividing Eq. (7.97) by A_{P}; note that index 'e' on a mass CV represents index P on a u-CV):

$$u_e^* = \tilde{u}_e^* - \frac{S_e}{A_P^u}(p_E - p_P)^{m-1} , \tag{7.102}$$

where $\tilde{u}_e^*$ is shorthand notation for

$$\tilde{u}^* = \frac{Q_P^u - Q_u^P - \sum_l A_l^u u_l^*}{A_P} . \tag{7.103}$$

Analogously, v_n^* can be expressed as:

$$v_n^* = \tilde{v}_n^* - \frac{S_n}{A_P^v}(p_N - p_P)^{m-1} . \tag{7.104}$$

The velocities u^* and v^* need to be corrected to enforce mass conservation. This is done, as outlined in Sect. 7.3.4, by correcting the pressure. The corrected velocities – the final values for the mth outer iteration – $u^m = u^* + u'$ and $v^m = v^* + v'$ are required to satisfy the linearized momentum equations, which is possible only if the pressure is corrected. So we write:

$$u_e^m = \tilde{u}_e^m - \frac{S_e}{A_P^u}(p_E - p_P)^m , \tag{7.105}$$

and

$$v_n^m = \tilde{v}_n^m - \frac{S_n}{A_P^v}(p_N - p_P)^m , \tag{7.106}$$

where $p^m = p^{m-1} + p'$ is the new pressure. The indices refer to the mass CV. The relation between velocity and pressure corrections is obtained by subtracting Eq. (7.102) from Eq. (7.105):

$$u_e' = \tilde{u}_e' - \frac{S_e}{A_P^u}(p_E' - p_P') , \tag{7.107}$$

where $\tilde{u}'$ is defined as (see Eq. (7.103)):

$$\tilde{u}_e' = \tilde{u}_e^m - \tilde{u}_e^* = -\frac{\sum_l A_l^u u_l'}{A_P} . \tag{7.108}$$

Analogously, one obtains:

$$v_n' = \tilde{v}_n' - \frac{S_n}{A_P^v}(p_N' - p_P') . \tag{7.109}$$

The corrected velocities are required to satisfy the continuity equation, so we substitute u^m and v^m into the expressions for mass fluxes, Eq. (7.83), and use Eq. (7.101):

$$(\rho S u')_e - (\rho S u')_w + (\rho S v')_n - (\rho S v')_s + \Delta \dot{m}_P^* = 0 . \tag{7.110}$$

Finally, substitution of the above expressions (7.107) and (7.109) for u' and v' into the continuity equation leads to the pressure-correction equation:

$$A_{\mathrm{P}}^{p} p'_{\mathrm{P}} + \sum_{l} A_{l}^{p} p'_{l} = -\Delta \dot{m}_{\mathrm{P}}^{*} - \Delta \dot{m}'_{\mathrm{P}} \ , \tag{7.111}$$

where the coefficients are:

$$\begin{aligned} A_{\mathrm{E}}^{p} &= -\left(\frac{\rho S^2}{A_{\mathrm{P}}^{u}}\right)_{\mathrm{e}} , \; A_{\mathrm{W}}^{p} = -\left(\frac{\rho S^2}{A_{\mathrm{P}}^{u}}\right)_{\mathrm{w}} , \\ A_{\mathrm{N}}^{p} &= -\left(\frac{\rho S^2}{A_{\mathrm{P}}^{v}}\right)_{\mathrm{n}} , \; A_{\mathrm{S}}^{p} = -\left(\frac{\rho S^2}{A_{\mathrm{P}}^{v}}\right)_{\mathrm{s}} , \\ A_{\mathrm{P}}^{p} &= -\sum_{l} A_{l}^{p} \ , \; l = \mathrm{E}, \mathrm{W}, \mathrm{N}, \mathrm{S} \ . \end{aligned} \tag{7.112}$$

The term $\Delta \dot{m}'_{\mathrm{P}}$ is analogous to $\Delta \dot{m}_{\mathrm{P}}^{*}$, with $\tilde{u}'$ and $\tilde{v}'$ replacing u^* and v^*, see Eqs. (7.101) and (7.110). Since the velocity corrections are not known prior to the solution of the pressure-correction equation, this term is usually neglected, as mentioned in the previous section. We then have the SIMPLE algorithm.

After the pressure-correction equation has been solved, the velocities and pressure are corrected. As noted in Sect. 7.3.4, if one tries to calculate steady flows using very large time steps, the momentum equations must be under-relaxed as described in Sect. 5.4.2, so only part of the pressure correction p' is added to p^{m-1}. Under-relaxation may also be required in unsteady calculations with large time steps.

The corrected velocities satisfy the continuity equation to the accuracy with which the pressure-correction equation is solved. However, they do not satisfy the non-linear momentum equation, so we have to begin another outer iteration. When both the continuity and momentum equations are satisfied to the desired tolerance, we can proceed to the next time level. To begin the iterations at the new time step, the solution at the previous time step provides the initial guess. This may be improved by use of extrapolation. For small time steps extrapolation is fairly accurate and saves a few iterations.

A computer code employing this algorithm is available via Internet; see Appendix for details. Some examples of its application and performance are presented below.

The above algorithm is easily modified to give the SIMPLEC method described in Sect. 7.3.4. The pressure-correction equation has the form (7.111), but A_{P}^{u} and A_{P}^{v} are replaced by $A_{\mathrm{P}}^{u} + \sum_l A_l^u$ and $A_{\mathrm{P}}^{v} + \sum_l A_l^v$, respectively. The extension to the PISO algorithm is also straightforward. The second pressure-correction equation has the same coefficient matrix as the first one, but the source term is now $-\Delta \dot{m}'_{\mathrm{P}}$. This term was neglected in the first pressure-correction equation, but it can now be calculated using the first velocity correction u'_i.

Discretizations of higher order are easily incorporated into the above solution strategy. The implementation of boundary conditions will be described after discussing the solution methods using colocated arrangement of variables.

7.5.2 Treatment of Pressure for Colocated Variables

It was mentioned earlier that a colocated arrangement of variables on a numerical grid creates problems which caused it to be out of favor until recently. Here, we shall first show why the problems occur and then present a cure.

We start by looking at a finite-difference scheme and the simple time-advance method presented in Sect. 7.3.2. There we derived the discrete Poisson equation for the pressure, which can be written:

$$\frac{\delta}{\delta x_i}\left(\frac{\delta p^n}{\delta x_i}\right) = \frac{\delta H_i^n}{\delta x_i} , \tag{7.113}$$

where H_i^n is the shorthand notation for the sum of the advective and viscous terms:

$$H_i^n = -\frac{\delta(\rho u_i u_j)^n}{\delta x_j} + \frac{\delta \tau_{ij}^n}{\delta x_j} \tag{7.114}$$

(summation on j is implied). The discretization scheme used to approximate the derivatives is not important in Eq. (7.113), that is why symbolic notation is used. Also, the equation is not specific to any grid arrangement.

Let us now look at the colocated arrangement shown in Fig. 7.5 and various difference schemes for the pressure gradient terms in the momentum equations and for the divergence in the continuity equation. We start by considering a forward difference scheme for pressure terms and a backward difference scheme for the continuity equation. Section 7.1.3 shows that this combination is energy conserving. For simplicity we assume that the grid is uniform with spacings Δx and Δy.
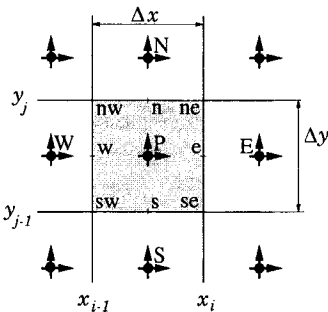
Fig. 7.5. Control volume in a colocated grid and notation used

By approximating the outer difference operator $\delta/\delta x_i$ in the pressure equation with the backward difference scheme, we obtain:

$$\frac{\left(\frac{\delta p^n}{\delta x}\right)_\mathrm{P} - \left(\frac{\delta p^n}{\delta x}\right)_\mathrm{W}}{\Delta x} + \frac{\left(\frac{\delta p^n}{\delta y}\right)_\mathrm{P} - \left(\frac{\delta p^n}{\delta y}\right)_\mathrm{S}}{\Delta y} = \frac{H^n_{x,\mathrm{P}} - H^n_{x,\mathrm{W}}}{\Delta x} + \frac{H^n_{y,\mathrm{P}} - H^n_{y,\mathrm{S}}}{\Delta y} \; . \tag{7.115}$$

Denoting the right hand side as Q^H_P and using the forward difference approximations for the pressure derivatives, we arrive at:

$$\frac{\frac{p^n_\mathrm{E} - p^n_\mathrm{P}}{\Delta x} - \frac{p^n_\mathrm{P} - p^n_\mathrm{W}}{\Delta x}}{\Delta x} + \frac{\frac{p^n_\mathrm{N} - p^n_\mathrm{P}}{\Delta y} - \frac{p^n_\mathrm{P} - p^n_\mathrm{S}}{\Delta y}}{\Delta y} = Q^H_\mathrm{P} \; . \tag{7.116}$$

The system of algebraic equations for the pressure then takes the form:

$$A^p_\mathrm{P} p^n_\mathrm{P} + \sum_l A^p_l p^n_l = -Q^H_\mathrm{P} \; , \quad l = \mathrm{E, W, N, S} \; , \tag{7.117}$$

where the coefficients are:

$$A^p_\mathrm{E} = A^p_\mathrm{W} = -\frac{1}{(\Delta x)^2} \; , \quad A^p_\mathrm{N} = A^p_\mathrm{S} = -\frac{1}{(\Delta y)^2} \; , \qquad A^p_\mathrm{P} = -\textstyle\sum_l A^p_l \; . \tag{7.118}$$

One can verify that the FV approach would reproduce equation (7.116) if the CV shown in Fig. 7.5 is used for both the momentum equations and the continuity equation, and if the following approximations are used: $u_\mathrm{e} = u_\mathrm{P}$, $p_\mathrm{e} = p_\mathrm{E}$; $v_\mathrm{n} = v_\mathrm{P}$, $p_\mathrm{n} = p_\mathrm{N}$; $u_\mathrm{w} = u_\mathrm{W}$, $p_\mathrm{w} = p_\mathrm{P}$; $v_\mathrm{s} = v_\mathrm{S}$, $p_\mathrm{s} = p_\mathrm{P}$.

The pressure or pressure-correction equation has the same form as the one obtained on a staggered grid with central difference approximations; this is because approximation of a second derivative by a product of forward and backward difference approximations for first derivatives gives the central difference approximation. However, the momentum equations suffer from use of a first order approximation to the major driving force term – the pressure gradient. It is better to use higher order approximations.

Now consider what happens if we choose central difference approximations for both the pressure gradient in the momentum equations and the divergence in the continuity equation. Approximating the outer difference operator in Eq. (7.113) by central differences, we obtain:

$$\frac{\left(\frac{\delta p^n}{\delta x}\right)_\mathrm{E} - \left(\frac{\delta p^n}{\delta x}\right)_\mathrm{W}}{2\Delta x} + \frac{\left(\frac{\delta p^n}{\delta y}\right)_\mathrm{N} - \left(\frac{\delta p^n}{\delta y}\right)_\mathrm{S}}{2\Delta y} =$$

$$\frac{H^n_{x,\mathrm{E}} - H^n_{x,\mathrm{W}}}{2\Delta x} + \frac{H^n_{y,\mathrm{N}} - H^n_{y,\mathrm{S}}}{2\Delta y} \ . \tag{7.119}$$

We again denote the right hand side as Q^H_P; however, this quantity is not the one obtained previously. Inserting the central difference approximations for pressure derivatives, we find:

$$\frac{\dfrac{p^n_\mathrm{EE} - p^n_\mathrm{P}}{2\Delta x} - \dfrac{p^n_\mathrm{P} - p^n_\mathrm{WW}}{2\Delta x}}{2\Delta x} + \frac{\dfrac{p^n_\mathrm{NN} - p^n_\mathrm{P}}{2\Delta y} - \dfrac{p^n_\mathrm{P} - p^n_\mathrm{SS}}{2\Delta y}}{2\Delta y} = Q^H_\mathrm{P} \ . \tag{7.120}$$

The system of algebraic equations for the pressure has the form:

$$A^p_\mathrm{P} p^n_\mathrm{P} + \sum_l A^p_l p^n_l = -Q^H_\mathrm{P} \ , \quad l = \mathrm{EE, WW, NN, SS} \tag{7.121}$$

where the coefficients are:

$$A^p_\mathrm{EE} = A^p_\mathrm{WW} = -\frac{1}{(2\Delta x)^2} \ , \quad A^p_\mathrm{NN} = A^p_\mathrm{SS} = -\frac{1}{(2\Delta y)^2} \ , \\ A^p_\mathrm{P} = -\textstyle\sum_l A^p_l \tag{7.122}$$

This equation has the same form as Eq. (7.117) but it involves nodes which are $2\Delta x$ apart! It is a discretized Poisson equation on a grid twice as coarse as the basic one but the equations split into four unconnected systems, one with i and j both even, one with i even and j odd, one with i odd and j even, and one with both odd. Each of these systems gives a different solution. For a flow with a uniform pressure field, the checkerboard pressure distribution shown in Fig. 7.6 satisfies these equations and could be produced. However, the pressure gradient is not affected and the velocity field may be smooth. There is also the possibility that one may not be able to obtain a converged steady-state solution.

A similar result is obtained with the finite volume approach if the CV face values of the fluxes are calculated by linear interpolation of the two neighbor nodes.
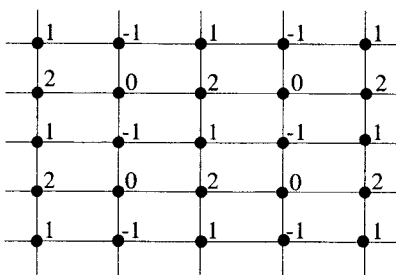
Fig. 7.6. Checkerboard pressure field, made of four superimposed uniform fields on 2Δ-spacing, which is interpreted by CDS as a uniform field

The source of the above problem may be traced to using $2\Delta x$ approximations to the first derivatives. Various cures have been proposed. In incompressible flows, the absolute pressure level is unimportant – only the differences matter. Unless the absolute value of the pressure is specified somewhere, the pressure equation is singular and has an infinite number of solutions, all differing by a constant. This makes a simple cure possible: filtering out the oscillations, as was done by van der Wijngaart (1990).

We shall present one approach to dealing with the pressure-velocity coupling on colocated grids that has found widespread use in complicated geometries and is simple and effective.

On staggered grids, central difference approximations are based on Δx differences. Can we do the same with the colocated arrangement? A Δx approximation of the outer first derivative in the pressure equation (7.113) has the form:

$$\frac{\left(\frac{\delta p^n}{\delta x}\right)_{\mathrm{e}} - \left(\frac{\delta p^n}{\delta x}\right)_{\mathrm{w}}}{\Delta x} + \frac{\left(\frac{\delta p^n}{\delta y}\right)_{\mathrm{n}} - \left(\frac{\delta p^n}{\delta y}\right)_{\mathrm{s}}}{\Delta y} =$$

$$\frac{H^n_{x,\mathrm{e}} - H^n_{x,\mathrm{w}}}{\Delta x} + \frac{H^n_{y,\mathrm{n}} - H^n_{y,\mathrm{s}}}{\Delta y} \quad . \tag{7.123}$$

The problem is that the values of pressure derivatives and quantities H are not available at cell face locations, so we have to use interpolation. Let us choose linear interpolation, which has the same accuracy as the CDS approximation of the derivatives. Also let the inner derivatives of the pressure in Eq. (7.113) be approximated by central differences. Linear interpolation of cell center derivatives leads to:

$$\left(\frac{\delta p^n}{\delta x}\right)_{\mathrm{e}} \approx \frac{1}{2}\left(\frac{p_{\mathrm{E}} - p_{\mathrm{W}}}{2\,\Delta x} + \frac{p_{\mathrm{EE}} - p_{\mathrm{P}}}{2\,\Delta x}\right) \, . \tag{7.124}$$

With this interpolation the pressure equation (7.120) is recovered.

We could evaluate the pressure derivatives at cell faces using central differences and Δx spacing as follows:

$$\left(\frac{\delta p^n}{\delta x}\right)_{\mathrm{e}} \approx \frac{p_{\mathrm{E}} - p_{\mathrm{P}}}{\Delta x} \, . \tag{7.125}$$

If this approximation is applied at all cell faces, we arrive at the following pressure equation (which is also valid on non-uniform grids):

$$\frac{\frac{p^n_{\mathrm{E}} - p^n_{\mathrm{P}}}{\Delta x} - \frac{p^n_{\mathrm{P}} - p^n_{\mathrm{W}}}{\Delta x}}{\Delta x} + \frac{\frac{p^n_{\mathrm{N}} - p^n_{\mathrm{P}}}{\Delta y} - \frac{p^n_{\mathrm{P}} - p^n_{\mathrm{S}}}{\Delta y}}{\Delta y} = Q^H_{\mathrm{P}} \, , \tag{7.126}$$

which is the same as Eq. (7.116), except that the right hand side is now obtained by interpolation:

$$Q_{\mathrm{P}}^{H} = \frac{\overline{(H_x^n)_{\mathrm{e}}} - \overline{(H_x^n)_{\mathrm{w}}}}{\Delta x} + \frac{\overline{(H_y^n)_{\mathrm{n}}} - \overline{(H_y^n)_{\mathrm{s}}}}{\Delta y} . \tag{7.127}$$

Use of this approximation eliminates the oscillation in the pressure field but, in order to accomplish this, we have introduced an inconsistency in the treatment of the pressure gradient in the momentum and pressure equations. Let us compare the two approximations. It is easy to show that the left hand sides of Eqs. (7.126) and (7.120) differ by:

$$R_{\mathrm{P}}^{p} = \frac{4\,p_{\mathrm{E}} + 4\,p_{\mathrm{W}} - 6\,p_{\mathrm{P}} - p_{\mathrm{EE}} - p_{\mathrm{WW}}}{4(\Delta x)^2} + \frac{4\,p_{\mathrm{N}} + 4\,p_{\mathrm{S}} - 6\,p_{\mathrm{P}} - p_{\mathrm{NN}} - p_{\mathrm{SS}}}{4(\Delta y)^2} , \tag{7.128}$$

which represents a central difference approximation to the fourth order pressure derivatives:

$$R_{\mathrm{P}}^{p} = -\frac{(\Delta x)^2}{4}\left(\frac{\partial^4 p}{\partial x^4}\right)_{\mathrm{P}} - \frac{(\Delta y)^2}{4}\left(\frac{\partial^4 p}{\partial y^4}\right)_{\mathrm{P}} . \tag{7.129}$$

Expression (7.128) is easily obtained by applying the standard CDS approximation of the second derivative twice, see Sect. 3.4.

This difference tends to zero as the grid is refined and the error introduced is of the same magnitude as the error in the basic discretization and so does not add significantly to the latter. However, the energy conserving property of the scheme is destroyed in this process, introducing the possibility of instability.

The above result was derived for second order CDS discretization and linear interpolation. A similar derivation can be constructed for any discretization scheme and interpolation. Let us see how the above idea translates into an implicit pressure-correction method using FV discretization.

7.5.3 SIMPLE Algorithm for a Colocated Variable Arrangement

Implicit solution of the momentum equations discretized with a colocated FV method follows the line of the previous section for the staggered arrangement. One has only to bear in mind that the CVs for all variables are the same. The pressures at the cell face centers, which are not nodal locations, have to be obtained by interpolation; linear interpolation is a suitable second order approximation, but higher order methods can be used. The gradients at the CV center, which are needed for the calculation of cell face velocities, can be obtained using Gauss' theorem. The pressure forces in the x and y direction are summed over all faces and divided by the cell volume to yield the corresponding mean pressure derivative, e.g.:

$$\left(\frac{\delta p}{\delta x}\right)_{\mathrm{P}} = \frac{Q_u^p}{\Delta\Omega} , \tag{7.130}$$

where Q_u^p stands for the sum of pressure forces in the x-direction over all CV faces, see Eq. (7.91). On Cartesian grids this reduces to the standard CDS approximation.

Solution of the linearized momentum equations produces u^* and v^*. For the discretized continuity equation, we need the cell face velocities which have to be calculated by interpolation; linear interpolation is the obvious choice. The pressure-correction equation of the SIMPLE algorithm can be derived following the lines of Sects. 7.3.4 and 7.5.1. The interpolated cell face velocities needed in the continuity equation involve interpolated pressure gradients, so their correction is proportional to the interpolated pressure correction gradient (see Eq. (7.46)):

$$u'_{\mathrm{e}} = -\overline{\left(\frac{\Delta\Omega}{A_{\mathrm{P}}^u}\frac{\delta p'}{\delta x}\right)_{\mathrm{e}}} . \tag{7.131}$$

On uniform grids, the pressure-correction equation derived using this expression for the cell face velocity corrections corresponds to Eq. (7.120). On non-uniform grids, the computational molecule of the pressure-correction equation involves the nodes P, E, W, N, S, EE, WW, NN and SS. As shown in the preceding section, this equation may have oscillatory solutions. Although the oscillations can be filtered out (see van der Wijngaart, 1990), the pressure-correction equation becomes complex on arbitrary grids and the convergence of the solution algorithm may be slow. A compact pressure-correction equation similar to the staggered grid equation can be obtained using the approach discussed in the preceding section. It is described below.

It was shown in the preceding section that the interpolated pressure gradients can be replaced by compact central-difference approximations at the cell faces. The interpolated cell face velocity is thus modified by the difference between the interpolated pressure gradient and the gradient calculated at the cell face:

$$u_{\mathrm{e}}^* = \overline{(u^*)}_{\mathrm{e}} - \Delta\Omega_{\mathrm{e}}\overline{\left(\frac{1}{A_{\mathrm{P}}^u}\right)}_{\mathrm{e}}\left[\left(\frac{\delta p^{m-1}}{\delta x}\right)_{\mathrm{e}} - \overline{\left(\frac{\delta p^{m-1}}{\delta x}\right)}_{\mathrm{e}}\right] . \tag{7.132}$$

An overbar denotes interpolation, and the volume centered around a cell face is defined by:

$$\Delta\Omega_{\mathrm{e}} = (x_{\mathrm{E}} - x_{\mathrm{P}})\,\Delta y .$$

for Cartesian grids.

This procedure adds a correction to the interpolated velocity that is proportional to the third derivative of the pressure multiplied by $(\Delta x)^2/4$; the fourth derivative for cell center results from applying the divergence operator.

In a second order scheme, the pressure derivative at the cell face is calculated using CDS, see Eq. (7.125). If the CDS approximation (7.125) is used on non-uniform grids, the cell-center pressure gradients should be interpolated with weights 1/2, since this approximation does not 'see' the grid non-uniformity.

The correction will be large if the pressure oscillates rapidly; the third derivative is then large and will activate the pressure-correction and smooth out the pressure.

The correction to the cell face velocity in the SIMPLE method is now:

$$u'_{\mathrm{e}} = -\Delta\Omega_{\mathrm{e}} \overline{\left(\frac{1}{A_{\mathrm{P}}^{u}}\right)_{\mathrm{e}}} \left(\frac{\delta p'}{\delta x}\right)_{\mathrm{e}} = -S_{\mathrm{e}} \overline{\left(\frac{1}{A_{\mathrm{P}}^{u}}\right)_{\mathrm{e}}} (p'_{\mathrm{E}} - p'_{\mathrm{P}}) \, , \tag{7.133}$$

with corresponding expressions at other cell faces. When these are inserted into the discretized continuity equation, the result is again the pressure-correction equation (7.111). The only difference is that the coefficients $1/A_{\mathrm{P}}^{u}$ and $1/A_{\mathrm{P}}^{v}$ at the cell faces are not the nodal values, as in the staggered arrangement, but are interpolated cell center values.

Since the correction term in Eq. (7.132) is multiplied by A_{P}^{u}, the value of the under-relaxation parameter contained in them may affect the converged cell face velocity. However, there is little reason for concern, since the difference in the two solutions obtained using different under-relaxation parameters is much smaller than the discretization error, as will be shown in the examples below. We also show that the implicit algorithm using colocated grids has the same convergence rate, dependence on under-relaxation factor, and computing cost as the staggered grid algorithm. Furthermore, the difference between solutions obtained with different variable arrangements is also much smaller than the discretization error.

We have derived the pressure-correction equation on colocated grids for second order approximations. The method can be adapted to approximations of higher order; it is important that the differentiation and interpolation be of the same order. For a description of a fourth order method, see Lilek and Perić (1995).

Why solve the momentum equations at colocated nodes, and then calculate the velocities at staggered locations rather than using the staggered arrangement in the first place? In fact, for Cartesian grids and explicit schemes there is not much incentive to use the colocated arrangement. However, for non-orthogonal or unstructured grids, complex geometries, and for multigrid solution methods the colocated arrangement becomes attractive. This issue is discussed in the next chapter.

7.6 Note on Pressure and Incompressibility

Suppose that we have a velocity field $\boldsymbol{v}^*$, which does not satisfy the continuity condition; for example, $\boldsymbol{v}^*$ may have been obtained by time-advancing the

Navier-Stokes equations without invoking continuity. We wish to create a new velocity field $\boldsymbol{v}$ which:

- satisfies continuity,
- is as close as possible to the original field $\boldsymbol{v}^*$.

Mathematically we can pose this problem as one of minimizing:

$$\tilde{R} = \frac{1}{2}\int_{\Omega} [\boldsymbol{v}(\boldsymbol{r}) - \boldsymbol{v}^*(\boldsymbol{r})]^2 \,\mathrm{d}\Omega \,, \tag{7.134}$$

where $\boldsymbol{r}$ is the position vector and Ω is the domain over which the velocity field is defined, subject to the continuity constraint

$$\boldsymbol{\nabla} \cdot \boldsymbol{v}(\boldsymbol{r}) = 0 \tag{7.135}$$

being satisfied everywhere in the field. The question of boundary conditions will be dealt with below.

This is a standard type of problem of the calculus of variations. A useful way of dealing with it is to introduce a Lagrange multiplier. The original problem (7.134) is replaced by the problem of minimizing:

$$R = \frac{1}{2}\int_{\Omega} [\boldsymbol{v}(\boldsymbol{r}) - \boldsymbol{v}^*(\boldsymbol{r})]^2 \,\mathrm{d}\Omega - \int_{\Omega} \lambda(\boldsymbol{r})\, \boldsymbol{\nabla} \cdot \boldsymbol{v}(\boldsymbol{r}) \,\mathrm{d}\Omega \,, \tag{7.136}$$

where λ is the Lagrange multiplier. The inclusion of the Lagrange multiplier term does not affect the minimum value since the constraint (7.135) requires it to be zero.

Suppose that the function that minimizes the functional R is $\boldsymbol{v}^+$; of course, $\boldsymbol{v}^+$ also satisfies (7.135). Thus:

$$R_{\mathrm{min}} = \frac{1}{2}\int_{\Omega} [\boldsymbol{v}^+(\boldsymbol{r}) - \boldsymbol{v}^*(\boldsymbol{r})]^2 \,\mathrm{d}\Omega \,. \tag{7.137}$$

If R_{min} is a true minimum, then any deviation from $\boldsymbol{v}^+$ must produce a second-order change in R. Thus suppose that:

$$\boldsymbol{v} = \boldsymbol{v}^+ + \delta\boldsymbol{v} \,, \tag{7.138}$$

where $\delta\boldsymbol{v}$ is small but arbitrary. When $\boldsymbol{v}$ is substituted into the expression (7.136), the result is $R_{\mathrm{min}} + \delta R$ where:

$$\delta R = \int_{\Omega} \delta\boldsymbol{v}(\boldsymbol{r}) \cdot [\boldsymbol{v}^+(\boldsymbol{r}) - \boldsymbol{v}^*(\boldsymbol{r})] \,\mathrm{d}\Omega - \int_{\Omega} \lambda(\boldsymbol{r}) \boldsymbol{\nabla} \cdot \delta\boldsymbol{v}(\boldsymbol{r}) \,\mathrm{d}\Omega \,. \tag{7.139}$$

We have dropped the term proportional to $(\delta\boldsymbol{v})^2$ as it is of second order. Now, integrating the last term by parts and applying Gauss's theorem, we obtain:

$$\delta R = \int_{\Omega} \delta \boldsymbol{v}(\boldsymbol{r})\cdot[\boldsymbol{v}^{+}(\boldsymbol{r})-\boldsymbol{v}^{*}(\boldsymbol{r})+\nabla\lambda(\boldsymbol{r})]\,\mathrm{d}\Omega+\int_{S}\lambda(\boldsymbol{r})\,\delta\boldsymbol{v}(\boldsymbol{r})\cdot\boldsymbol{n}\,\mathrm{d}S\;. \quad (7.140)$$

On the parts of the domain surface on which a boundary condition on $\boldsymbol{v}$ is given (walls, inflow), it is presumed that both $\boldsymbol{v}$ and $\boldsymbol{v}^+$ satisfy the given condition so $\delta\boldsymbol{v}$ is zero there. These portions of the boundaries make no contribution to the surface integral in Eq. (7.140) so no condition on λ is required on them; however, a condition will be developed below. On those parts of the boundary where other types of boundary conditions are given (symmetry planes, outflows) $\delta\boldsymbol{v}$ is not necessarily zero; to make the surface integral vanish, we need to require that $\lambda = 0$ on these portions of the boundary.

If δR is to vanish for arbitrary $\delta\boldsymbol{v}$, we must require that the volume integral in Eq. (7.140) also vanishes, i.e.:

$$\boldsymbol{v}^{+}(\boldsymbol{r}) - \boldsymbol{v}^{*}(\boldsymbol{r}) + \nabla\lambda(\boldsymbol{r}) = 0\;. \quad (7.141)$$

Finally, we recall that $\boldsymbol{v}^+(\boldsymbol{r})$ must satisfy the continuity equation (7.135). Taking the divergence of Eq . (7.141) and applying this condition, we find:

$$\nabla^2\lambda(\boldsymbol{r}) = \nabla\cdot\boldsymbol{v}^{*}(\boldsymbol{r})\;, \quad (7.142)$$

which is a Poisson equation for $\lambda(\boldsymbol{r})$. On those portions of the boundary on which boundary conditions are given on $\boldsymbol{v}$, $\boldsymbol{v}^+ = \boldsymbol{v}^*$. Equation (7.141) shows that in this case $\nabla\lambda(\boldsymbol{r}) = 0$ and we have a boundary condition on λ.

If Eq. (7.142) and the boundary conditions are satisfied, the velocity field will be divergence free. It is also useful to note that this entire exercise can be repeated with the continuous operators replaced by discrete ones.

Once the Poisson equation in solved, the corrected velocity field is obtained from Eq. (7.141) written in the form:

$$\boldsymbol{v}^{+}(\boldsymbol{r}) = \boldsymbol{v}^{*}(\boldsymbol{r}) - \nabla\lambda(\boldsymbol{r})\;. \quad (7.143)$$

This shows that the Lagrange multiplier $\lambda(\boldsymbol{r})$ essentially plays the role of the pressure and again shows that, in incompressible flows, the function of the pressure is to allow continuity to be satisfied.

7.7 Boundary Conditions for the Navier-Stokes Equations

Everything that has been said about boundary conditions in Chaps. 3 and 4 for the generic conservation equation applies to the momentum equations. Some special features will be addressed in this section.

At a wall the no-slip boundary condition applies, i.e. the velocity of the fluid is equal to the wall velocity, a Dirichlet boundary condition. However, there is another condition that can be directly imposed in a FV method; the normal viscous stress is zero at a wall. This follows from the continuity equation, e.g. for a wall at $y = 0$ (see Fig. 7.7):

$$\left(\frac{\partial u}{\partial x}\right)_{\text{wall}} = 0 \Rightarrow \left(\frac{\partial v}{\partial y}\right)_{\text{wall}} = 0 \Rightarrow \tau_{yy} = 2\mu\left(\frac{\partial v}{\partial y}\right)_{\text{wall}} = 0 \,. \quad (7.144)$$

Therefore, the diffusive flux in the v equation at the south boundary is:

$$F_{\text{s}}^{\text{d}} = \int_{S_{\text{s}}} \tau_{yy}\,\mathrm{d}S = 0 \,. \quad (7.145)$$

This should be implemented directly, rather than using only the condition that $v = 0$ at the wall. Since $v_{\text{P}} \neq 0$, we would obtain a non-zero derivative in the discretized flux expression if this were not done; $v = 0$ is used as a boundary condition in the continuity equation. The shear stress can be calculated by using a one-sided approximation of the derivative $\partial u/\partial y$; one possible approximation is (for the u equation and the situation from Fig. 7.7):

$$F_{\text{s}}^{\text{d}} = \int_{S_{\text{s}}} \tau_{xy}\,\mathrm{d}S = \int_{S_{\text{s}}} \mu\frac{\partial u}{\partial y}\,\mathrm{d}S \approx \mu_{\text{s}} S_{\text{s}} \frac{u_{\text{P}} - u_{\text{S}}}{y_{\text{P}} - y_{\text{S}}} \,. \quad (7.146)$$
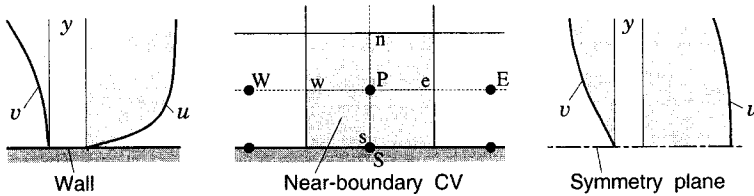
Fig. 7.7. On the boundary conditions at a wall and a symmetry plane

At a symmetry plane we have the opposite situation: the shear stress is zero, but the normal stress is not, since (for the situation from Fig. 7.7):

$$\left(\frac{\partial u}{\partial y}\right)_{\text{sym}} = 0 \,; \quad \left(\frac{\partial v}{\partial y}\right)_{\text{sym}} \neq 0 \,. \quad (7.147)$$

The diffusive flux in the u equation is zero, and the diffusive flux in the v equation can be approximated as:

$$F_{\text{s}}^{\text{d}} = \int_{S_{\text{s}}} \tau_{yy}\,\mathrm{d}S = \int_{S_{\text{s}}} 2\mu\frac{\partial v}{\partial y}\,\mathrm{d}S \approx \mu_{\text{s}} S_{\text{s}} \frac{v_{\text{P}} - v_{\text{S}}}{y_{\text{P}} - y_{\text{S}}} \quad (7.148)$$

where $v_{\text{S}} = 0$ applies.

In a FV method using a staggered grid, the pressure is not required at boundaries (except if the pressure at a boundary is specified; this is handled in Chap. 10). This is due to the fact that the nearest CV for the velocity component normal to the boundary extends only up to the center of the scalar

CV, where the pressure is calculated. When colocated arrangement is used, all CVs extend to the boundary and we need the boundary pressure to calculate the pressure forces in the momentum equations. We have to use extrapolation to obtain pressure at the boundaries. In most cases, linear extrapolation is sufficiently accurate for a second order method. However, there are cases in which a large pressure gradient near a wall is needed in the equation for the normal velocity component to balance a body force (buoyancy, centrifugal force etc.). If the pressure extrapolation is not accurate, this condition may not be satisfied and large normal velocities near the boundary may result. This can be avoided by calculating the normal velocity component for the first CV from the continuity equation, by adjusting the pressure extrapolation, or by local grid refinement.

The boundary conditions for the pressure-correction equation also deserve some attention. When the mass flux through a boundary is prescribed, the mass flux correction in the pressure correction equation is also zero there. This condition should be directly implemented in the continuity equation when deriving the pressure-correction equation. It is equivalent to specifying a Neumann boundary condition (zero gradient) for the pressure correction.

At the outlet, if the inlet mass fluxes are given, extrapolation of the velocity to the boundary (zero gradient, e.g. $u_{\rm E} = u_{\rm P}$) can usually be used for steady flows when the outflow boundary is far from the region of interest and the Reynolds number is large. The extrapolated velocity is then corrected to give exactly the same total mass flux as at inlet (this cannot be guaranteed by any extrapolation). The corrected velocities are then considered as prescribed for the following outer iteration and the mass flux correction at the outflow boundary is set to zero in the continuity equation. This leads to the pressure-correction equation having Neumann conditions on all boundaries and makes it singular. To make the solution unique, one usually takes the pressure at one point to be fixed, so the pressure correction calculated at that point is subtracted from all the corrected pressures. Another choice is to set the mean pressure to some value, say zero.

Another case is obtained when the pressure difference between the inlet and outlet boundaries is specified. Then the velocities at these boundaries cannot be specified – they have to be computed so that the pressure loss is the specified value. This can be implemented in several ways. In any case the boundary velocity has to be extrapolated from the inner nodes (in a manner similar to the interpolation for cell faces in a colocated arrangement) and then corrected. An example of how specified static pressure can be handled is given in Chap. 10.

7.8 Examples

In this section we shall demonstrate the accuracy and efficiency of the implicit SIMPLE method for steady incompressible flows on staggered and colocated

grids. As test cases we choose two flows in square enclosures; one flow is driven by a moving lid and the other by buoyancy. The geometry and boundary conditions are shown schematically in Fig. 7.8. Both test cases have been used by many authors and accurate solutions are available in the literature; see Ghia et al. (1982) and Hortmann et al. (1990).
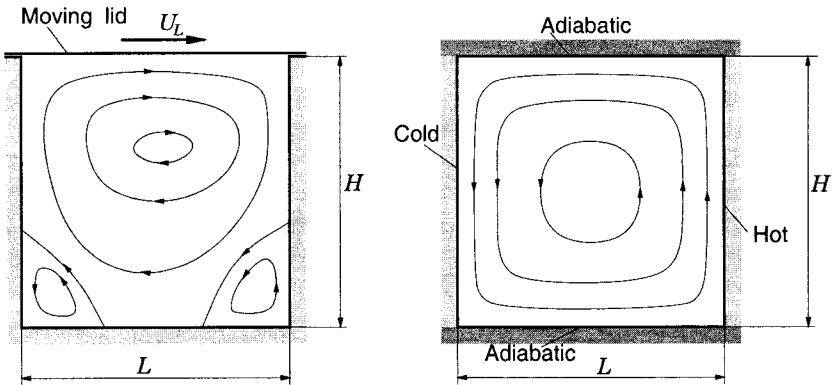
Fig. 7.8. Geometry and boundary conditions for 2D flow test cases: lid-driven (left) and buoyancy-driven (right) cavity flows
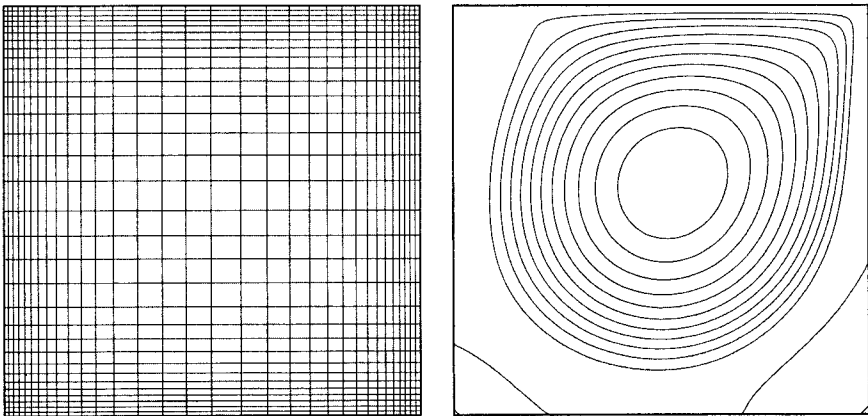
Fig. 7.9. A non-uniform grid with 32 × 32 CV used to solve the cavity flow problems (left) and the streamlines of the lid-driven cavity flow at Re = 1000 (right), calculated on a 128 × 128 CV non-uniform grid (the mass flow between any two adjacent streamlines is constant)

We first consider the lid-driven cavity flow. The moving lid creates a strong vortex and a sequence of weaker vortices in the lower two corners.

A non-uniform grid and the streamlines for the Reynolds number, based on cavity height H and lid velocity U_L, $\mathrm{Re} = U_L H/\nu = 1000$ are shown in Fig. 7.9.

We first look at the estimation of iteration convergence errors. Several methods were presented in Sect. 5.7.

First, an accurate solution was obtained by iterating until the residual norm became negligibly small (of the order of the round-off error in double precision). Then the calculation was repeated and the convergence error was computed as the difference between the converged solution obtained earlier and the intermediate solution.

Figure 7.10 shows the norm of the convergence error, the estimate obtained from Eqs. (5.86) or (5.93), the difference between two iterates, and the residual on a 32 × 32 CV grid with under-relaxation factors 0.7 for velocity and 0.3 for pressure. Since the algorithm needs many iterations to converge, the eigenvalues required by the error estimator were averaged over the latest 50 iterations. The fields were initiated by interpolating the solution on the next coarser grid, which is why the initial error is relatively low.
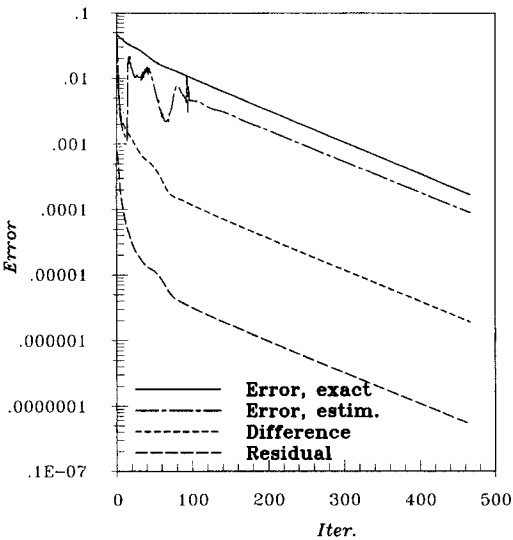
Fig. 7.10. Comparison of the norms of the exact and estimated convergence error for the SIMPLE method, the difference between two iterates and the residual for the solution of lid-driven cavity flow at $\mathrm{Re} = 10^3$ on a 32 × 32 CV grid, with CDS discretization

This figure shows that the error estimation technique gives good results for the non-linear flow problem. The estimate is not good at the beginning of the solution process, where the error is large. Using the absolute level of either the difference between two iterates or the residuals is not a reliable measure of the convergence error. These quantities do decrease at the same rate as the error, but need to be normalized properly to represent the convergence error quantitatively. Also, they fall very rapidly initially, while the error reduction is much slower. However, after a while all curves become nearly parallel and

if one knows roughly the order of the initial error (it is the solution itself if one starts with a zero initial field), then a reliable criterion for stopping the iterations is the reduction of the norm of either the difference between two iterates or the residual by a certain factor, say three or four orders of magnitude. Results similar to those shown in Fig. 7.10 are obtained on other grids and for other flow problems.

We turn next to the estimation of discretization errors. We performed computation on six grids using CDS discretization; the coarsest had 8 × 8 CV and the finest had 256 × 256 CV. Both uniform and non-uniform colocated grids were used. The strength of the primary vortex, ψ_{min}, which is the mass flow rate between the vortex center and the boundary, and the strength of the larger secondary vortex, ψ_{max}, were compared on all grids. Figure 7.11 shows the computed vortex strengths as the grid is refined. Results on the four finest grids show monotone convergence of both quantities towards the grid-independent solution. The results on the non-uniform grids are obviously more accurate.
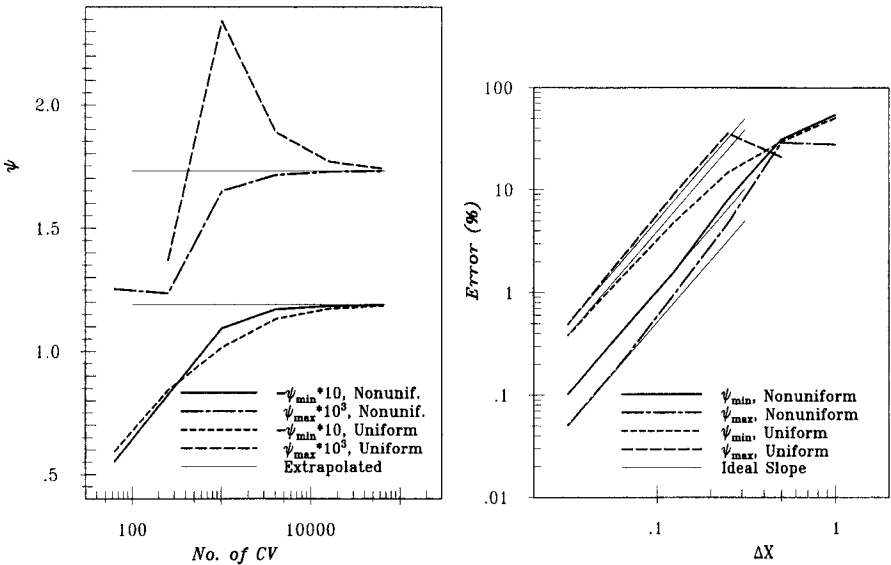
Fig. 7.11. Left: convergence of the strength of the primary (ψ_{min}) and secondary (ψ_{max}) vortex in a lid-driven cavity flow at Re = 1000 (calculation using CDS and both uniform and non-uniform grids); Right: errors in ψ_{min} and ψ_{max} as a function of the mesh spacing (normalized by the spacing on the coarsest grid)

In order to enable quantitative error estimation, the grid-independent solution was estimated using the results obtained on the two finest grids and Richardson extrapolation (see Sect. 3.9). These values are: $\psi_{\text{min}} = -0.1189$ and $\psi_{\text{max}} = 0.00173$. By subtracting results on a given grid from the reference

solution, an error estimate is obtained. The errors are plotted against grid size in Fig. 7.11. For both quantities on both uniform and non-uniform grids, the error reduction expected of a second order method is obtained. The errors are lower on non-uniform grids, especially for $\psi_{\max}$; since the secondary vortex is confined to a corner, the non-uniform grid, which is much finer there, yields higher accuracy.

Calculations on uniform colocated grids were also performed using three other discretization schemes: first order UDS, and cubic (fourth order) interpolation of the convective fluxes and midpoint rule integration, and fourth order Simpson's rule integration and cubic polynomial interpolation. Comparison of the profiles of the horizontal velocity component in the vertical symmetry plane, obtained with the four discretizations on grids ranging from 10×10 CV to 160×160 CV, are shown in Fig. 7.12. The first order scheme is so inaccurate that the solution on the finest grid is still too far from the grid-independent solution. The second order CDS shows monotone convergence; the ratio of errors on consecutive grids is a factor of four. Interpolation by a cubic polynomial increases the accuracy from the third grid onwards; on the first two grids, there are oscillations in the solution. Midpoint rule integration is used in both cases (i.e., the integrals are approximated by the product of the variable value at the cell face center and the cell face area). Finally, the use of Simpson's rule integration and cubic interpolation (a true fourth order method) gives the highest accuracy, except on the first two grids; the velocity profiles can hardly be distinguished from each other for grids 40 $\times$ 40 and finer.

The quantitative analysis of the accuracy of different schemes is presented in Fig. 7.13: it shows the convergence of the strength of the primary eddy towards the grid-independent solution and the estimated discretization errors. Second and fourth order schemes converge rapidly, while the first order UDS has a large error even on the finest grid (14.4%). Second order CDS and the mixed scheme produce the same estimate of the grid-independent value when Richardson extrapolation is applied (to within the convergence accuracy of the iterative solution method, which was between 0.01 and 0.1%). The plot of discretization error versus mesh spacing in Fig. 7.13 shows that the mixed scheme and the standard CDS closely follow the theoretical error reduction path for second order schemes. The UDS approaches the theoretical convergence rate as the grid is refined, but is more accurate than expected on coarse grids. Finally, fourth order CDS shows the expected error reduction from second to third grid; the discretization error then becomes comparable to the iteration convergence error, so that on finer grids the error is reduced less than expected. In order to obtain the theoretical error reduction rate on all grids, the iterative convergence error should have been at least an order of magnitude lower; on the finest grid this would mean iterating practically to the round-off level, which was not done because it would increase the computing time considerably.
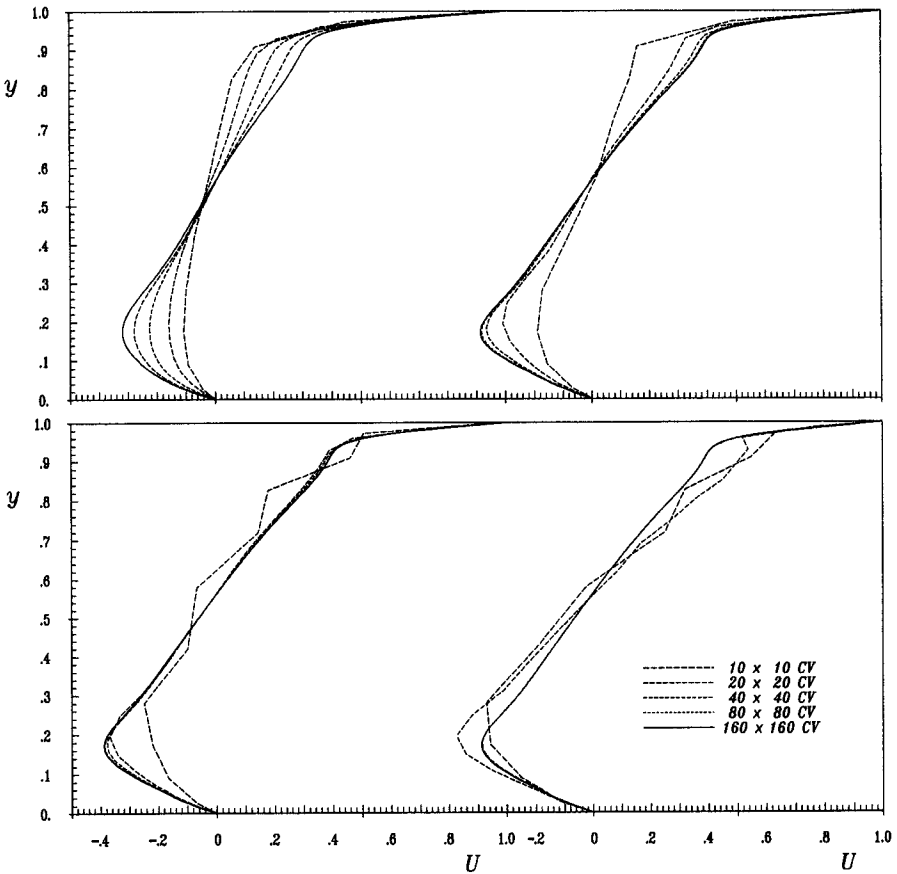
Fig. 7.12. Velocity profiles at the vertical centerline of the lid-driven cavity at Re = 1000, calculated on various grids using: midpoint rule and UDS (upper left), CDS (upper right) and cubic polynomial (lower left); Simpson's rule and cubic polynomials (lower right)

For all of the methods used, the grids had to be refined to 80 $\times$ 80 CV in order to judge the accuracy with confidence. If errors of the order of 1% are acceptable, then using CDS is the most efficient method (it is the simplest to implement and needs less computing time per iteration). Higher order methods are effective if small discretization errors are required (below 0.1%). On Cartesian grids, the fourth order scheme increases the memory requirement by about 30% and computing time per outer iteration by a factor of two; on irregular grids the cost increase would be much higher. The number of required iterations is roughly the same for all schemes (Lilek and Perić, 1995).
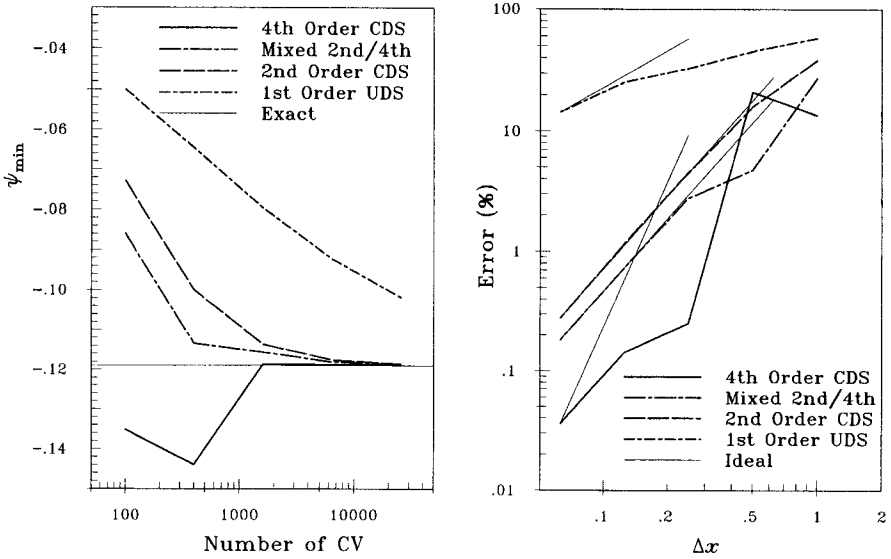
Fig. 7.13. Convergence of the strength of the primary vortex in a lid-driven cavity flow at Re = 1000 (left) and errors in $\psi_{\min}$ as functions of the mesh spacing (right); calculations on uniform grids from 10 × 10 CV to 160 × 160 CV using four different discretization schemes (the mixed scheme uses fourth order interpolation and second order integration)

We next investigate the difference between solutions obtained on uniform colocated and staggered grids using CDS discretization. Since the velocity nodes are at different locations on the two grids, staggered values were linearly interpolated to the cell centers (a higher-order interpolation would have been better but linear interpolation is good enough). The average difference is for each variable ($\phi = (u, v, p)$) determined as:

$$\epsilon = \frac{\sum_{i=1}^{N} |\phi_i^{\text{stag}} - \phi_i^{\text{col}}|}{N} , \tag{7.149}$$

where N is the number of CVs. For both u and v, ϵ is 1.2% on a grid with 10 × 10 CVs and 0.05% on a grid with 80 × 80 CVs. The difference is much smaller than the discretization errors on these grids (about 20% on 10 × 10 CV grid, about 1% on a grid with 80 × 80 CV, see Fig. 7.11). The differences in pressure were somewhat smaller.

Convergence properties of the SIMPLE method using CDS and staggered or colocated grids are investigated next. We first look at the effect of the under-relaxation parameter for pressure, α_p, see Eq. (7.45), on the convergence using various under-relaxation parameters for the velocity. Figure 7.14 shows the numbers of outer iterations required to reduce the residual level in

all equations three orders of magnitude using various combinations of under-relaxation parameters and a 32 × 32 CV uniform grid.
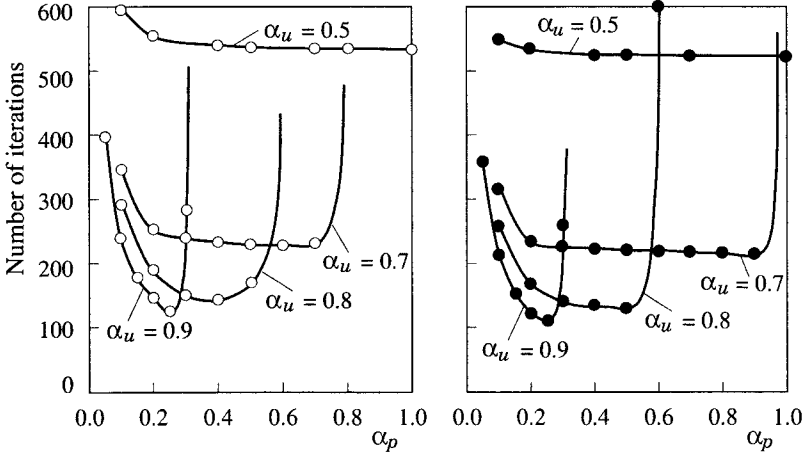
Fig. 7.14. Numbers of outer iterations required to reduce the residual level in all equations three orders of magnitude using various combinations of under-relaxation parameters and a 32 × 32 CV uniform grid with staggered (left) and colocated (right) arrangement of variables (lid-driven cavity flow at Re = 1000)

This figure shows that the dependence on the under-relaxation parameter α_p is almost the same for the two types of grids; the range of good values is somewhat wider for the colocated grid, but overall the behavior of both methods is similar. When the velocity is more strongly under-relaxed, we can use any value of α_p between 0.1 and 1.0, but the method converges slowly. For larger values of α_u, the convergence is faster but the useful range of α_p is restricted.

Patankar (1980) suggested using $\alpha_u = 0.5$ and $\alpha_p = 0.8$ in the SIMPLE method. We see from Fig. 7.14 that this is not optimum. The value of α_p suggested by Eq. 7.50 is nearly optimum; $\alpha_p = 1.1 - \alpha_u$ gives the best results for this flow and yields an improvement by about a factor of five over Patankar's recommendation.

In Fig. 7.15 we show the effect of the under-relaxation factor for velocity, α_u, on convergence rate for both staggered and colocated arrangements when optimum value of α_p is used, for two grids. We see that the method behaves in the same way for both grids. The dependence on α_u is stronger on the refined grid.

We next investigate the influence of the under-relaxation parameter for velocity on the solution for colocated grids. We choose $\alpha_u = 0.9$ for one case and $\alpha_u = 0.5$ for the other. The difference in the solutions (after residual levels were reduced five orders of magnitude, to exclude convergence errors)
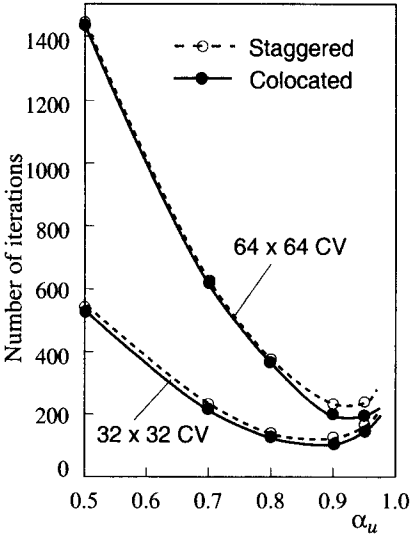
Fig. 7.15. Numbers of outer iterations required to reduce the residual level in all equations three orders of magnitude, as a function of the under-relaxation factor α_u (lid-driven cavity flow at Re = 1000)

was evaluated using the expression (7.149). It is 0.5% on the 8 × 8 CV grid and 0.002% on the 128 × 128 CV grid. These differences are two orders of magnitude smaller than the discretization errors on corresponding grids (53% and 0.4%, respectively) and can be neglected.
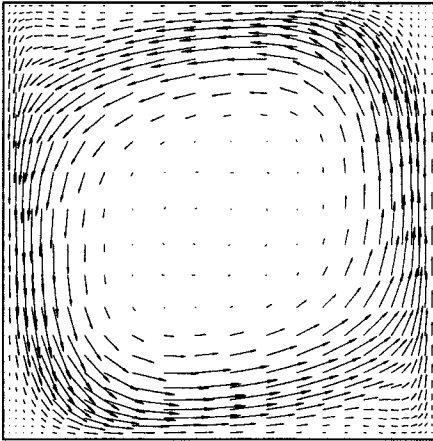
Fig. 7.16. Velocity vectors in buoyancy-driven cavity flow at the Rayleigh number Ra = 10^5 and Prandtl number Pr = 0.1

We next consider 2D buoyancy-driven flow in a square cavity as shown in Figs. 7.16 and 7.17. The cold and hot walls are isothermal. The heated fluid is rising along the hot wall, while cooled fluid is falling along the cold wall. The Prandtl number is 0.1, and the temperature difference and other fluid properties are chosen such that the Rayleigh number is
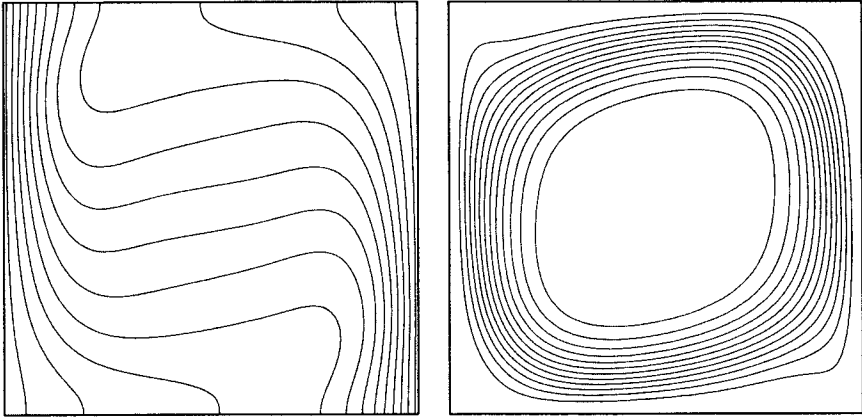
Fig. 7.17. Isotherms (left) and streamlines (right) in buoyancy-driven cavity flow at the Rayleigh number Ra = 10^5 and Prandtl number Pr = 0.1 (temperature difference between any two adjacent isotherms and mass flow rate between any two streamlines are same)

$$\mathrm{Ra} = \frac{\rho^2 g\beta(T_{\mathrm{hot}} - T_{\mathrm{cold}})H^3}{\mu^2}\,\mathrm{Pr} = 10^5 \; . \tag{7.150}$$

Velocity vectors, isotherms and streamlines are shown in Figs. 7.16 and 7.17. The flow structure depends strongly on the Prandtl number. A large core of almost stagnant, stably stratified fluid is formed in the central region of cavity. It is to be expected that non-uniform grids will give more accurate results than uniform grids. This is indeed so. Figure 7.18 shows total heat flux through the isothermal walls as a function of grid fineness for both uniform and non-uniform grids. Richardson extrapolation yields the same estimate of the grid-independent value to the five significant digits when applied to the results of two finest levels for both grid types. This estimate is $Q = 0.039248$, which, when normalized by the heat flux for pure heat conduction, $Q_{\mathrm{cond}} = 0.01$, gives the Nusselt number Nu = 3.9248. By subtracting the solutions on all grids from the estimated grid-independent solution, we obtain an estimate of the discretization error. The errors are plotted for both heat flux and the strength of the eddy against normalized mesh spacing in Fig. 7.18.

All errors tend asymptotically to the slope expected for second order schemes. The error in the heat flux is much smaller on the non-uniform than on uniform grid (more than an order of magnitude), while the error in eddy strength is smaller on a uniform grid. This rather unexpected finding can be explained as follows: the fine grid near walls increases accuracy of the heat transfer calculation, whereas coarse grid in the middle decreases accuracy of the representation of velocity profiles, which define the mass flow rate. However, mass flow rate error is rather small on both grids; e.g. for 64 $\times$ 64 CV, it is 0.03% on a uniform and 0.3% on a non-uniform grid.
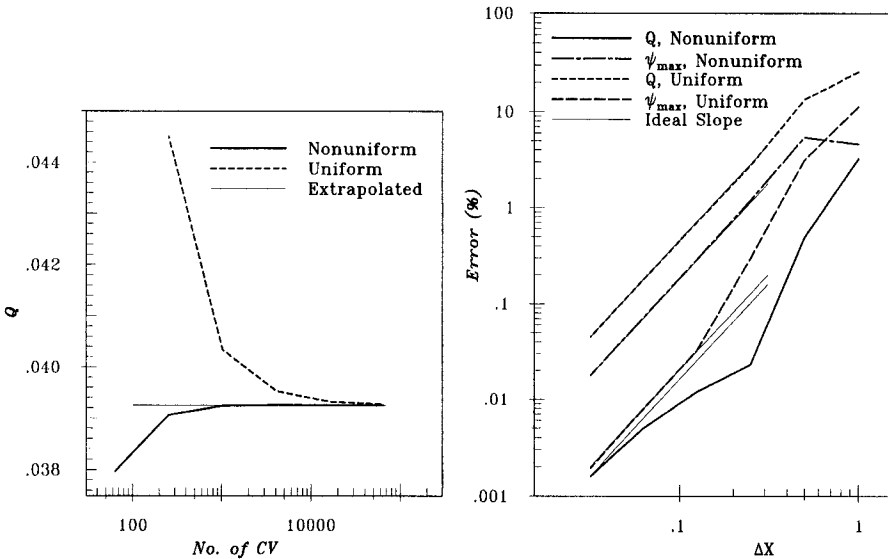
Fig. 7.18. Heat flux through isothermal walls, Q, (left) and the errors in the calculated heat flux and strength of the eddy (right) in a buoyancy-driven cavity flow at the Rayleigh number Ra $= 10^5$ and Prandtl number Pr $= 0.1$, as a function of grid fineness (calculation using CDS and both uniform and non-uniform grids)

We also checked the effect of the under-relaxation factor for the velocity on the solution in this case. As expected, it is much smaller than in the case of lid-driven cavity, since the pressure variation is much smoother here. On the coarsest grid (8×8 CV), the difference between solutions using $\alpha_u = 0.9$ and $\alpha_u = 0.5$ is about 0.23%, while on the grid with 128×128 CV it is below 0.0008%.

The efficiency of calculating steady incompressible flows using implicit methods based on SIMPLE algorithm and colocated grids can be substantially improved using multigrid method for outer iterations. This will be demonstrated for the two test cases studied here in Chap. 11.

8. Complex Geometries

Most flows in engineering practice involve complex geometries which are not readily fit with Cartesian grids. Although the principles of discretization and solution methods for algebraic systems described earlier may be used, many modifications are required. The properties of the solution algorithm depend on the choices of the grid and of the vector and tensor components, and the arrangement of variables on the grid. These issues are discussed in this chapter.

8.1 The Choice of Grid

When the geometry is regular (e.g. rectangular or circular), choosing the grid is simple: the grid lines usually follow the coordinate directions. In complicated geometries, the choice is not at all trivial. The grid is subject to constraints imposed by the discretization method. If the algorithm is designed for curvilinear orthogonal grids, non-orthogonal grids cannot be used; if the CVs are required to be quadrilaterals or hexahedra, grids consisting of triangles and tetrahedra cannot be used, etc. When the geometry is complex and the constraints cannot be fulfilled, compromises have to be made.

8.1.1 Stepwise Approximation Using Regular Grids

The simplest approach uses orthogonal grids (Cartesian or polar-cylindrical). In order to apply such a grid to solution domains with inclined or curved boundaries, the boundaries have to be approximated by staircase-like steps. This approach has been used, but it raises two kinds of problems:

- The number of grid points (or CVs) per grid line is not constant, as it is in a fully regular grid. This requires either indirect addressing, or special arrays have to be created that limit the index range on each line. The computer code may need to be changed for each new problem.
- The steps at the boundary introduce errors into the solution, especially when the grid is coarse. The treatment of the boundary conditions at stepwise walls also requires special attention.

An example of such a grid is shown in Fig. 8.1. This approach is a last resort, to be used when an existing solution method cannot be quickly adapted to a grid that fits boundary better. It is not recommended, except when the solution algorithm allows local grid refinement near the wall (see Chap. 11 for details of local grid refinement methods). An example is the large eddy simulation of flow over a wall-mounted hemisphere by Manhart and Wengle (1994).
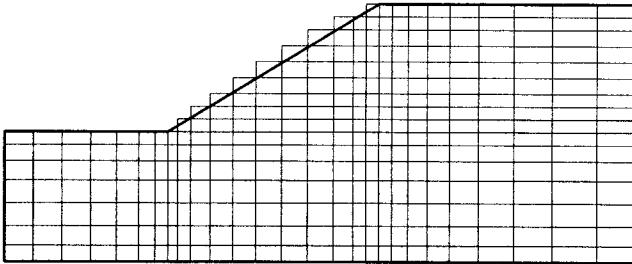
Fig. 8.1. An example of a grid using stepwise approximation of an inclined boundary

8.1.2 Overlapping Grids

Some authors suggest use of a set of regular grids to cover irregular solution domains. One can combine rectangular, cylindrical, spherical or non-orthogonal grids near bodies with Cartesian grids in the rest of the solution domain. An example is shown in Fig. 2.4. The disadvantage of this approach is that the programming and coupling of the grids can be complicated. The computation is usually sequential; the solution method is applied on one grid after another, the interpolated solution from one grid providing the boundary conditions for the next iteration on adjacent grids. It is also difficult to maintain conservation at the interfaces, and the interpolation process may introduce errors or convergence problems if the solution exhibits strong variation near the interface.

This method has also some attractive features. It allows – without additional difficulty – calculation of flows around bodies which move relative to the environment or each other. Each grid is attached to one reference frame, including some which move with the bodies. In such a case, the overlap region changes with time and has to be determined (together with the interpolation factors) after each time step. The grid has to be recalculated after each time step in any problem containing a moving body no matter what method is used, so this is not a drawback. The only additional effort is the interpolation from one reference frame to the other at the interfaces.

Grids of this kind are called *Chimera grids* in the literature (the Chimera is a mythological creature with lion's head, goat's body, and snake's tail). Examples of the use of overlapping grids are found in papers by Hinatsu and Ferziger (1991), Perng and Street (1991), Tu and Fuchs (1992), and Hubbard and Chen (1994,1995), among others.

8.1.3 Boundary-Fitted Non-Orthogonal Grids

Boundary-fitted non-orthogonal grids are most often used to calculate flows in complex geometries (most commercial codes use such grids). They can be structured, block-structured, or unstructured. The advantage of such grids is that they can be adapted to any geometry, and that optimum properties are easier to achieve than with orthogonal curvilinear grids. Since the grid lines follow the boundaries, the boundary conditions are more easily implemented than with stepwise approximation of curved boundaries. The grid can also be adapted to the flow, i.e. one set of grid lines can be chosen to follow the streamlines (which enhances the accuracy) and the spacing can be made smaller in regions of strong variable variation, especially if block-structured or unstructured grids are used.

Non-orthogonal grids have also several disadvantages. The transformed equations contain more terms thereby increasing both the difficulty of programming and the cost of solving the equations, the grid non-orthogonality may cause unphysical solutions and the arrangement of variables on the grid affects the accuracy and efficiency of the algorithm. These issues are discussed further below.

In the remainder of this book we shall assume that the grid is non-orthogonal. The principles of discretization and solution methods which we shall present are valid for orthogonal grids as well, since they can be viewed as a special case of a non-orthogonal grid. We shall also discuss the treatment of block-structured grids.

8.2 Grid Generation

The generation of grids for complex geometries is an issue which requires too much space to be dealt with in great detail here. We shall present only some basic ideas and the properties that a grid should have. More details about various methods of grid generation can be found in books and conference proceedings devoted to this topic, e.g. Thompson et al. (1985) and Arcilla et al. (1991).

Even though necessity demands that the grid be non-orthogonal, it is important to make it as nearly orthogonal as possible. In FV methods orthogonality of grid lines at CV vertices is unimportant – it is the angle between the cell face surface normal vector and the line connecting the CV centers on

either side of it that matters. Thus, a 2D grid made of equilateral triangles is equivalent to an orthogonal grid, since lines connecting cell centers are orthogonal to cell faces. This will be discussed further in Sect. 8.6.2.

Cell topology is also important. If the midpoint rule integral approximation, linear interpolation, and central differences are used to discretize the equations, then the accuracy will be higher if the CVs are quadrilaterals in 2D and hexahedra in 3D, than if we use triangles and tetrahedra, respectively. The reason is that parts of the errors made at opposite cell faces when discretizing diffusion terms cancel partially (if cell faces are parallel and of equal area, they cancel completely) on quadrilateral and hexahedral CVs. To obtain the same accuracy on triangles and tetrahedra, more sophisticated interpolation and difference approximations must be used. Especially near solid boundaries it is desirable to have quadrilaterals or hexahedra, since all quantities vary substantially there and accuracy is especially important in this region.

Accuracy is also improved if one set of grid lines closely follows the streamlines of the flow, especially for the convective terms. This cannot be achieved if triangles and/or tetrahedra are used, but is possible with quadrilaterals and hexahedra.

Non-uniform grids are the rule rather than exception when complex geometries are treated. The ratio of the sizes of adjacent cells should be kept under control, as accuracy is adversely affected if it is large. Especially when block-structured grids are used, one should take care that the cells are of nearly equal size near block interfaces; a factor of two variation should be the maximum. An experienced user may know where strong variation of velocity, pressure, temperature, etc. can be expected; the grid should be fine in these regions since the errors are most likely to be large there. However, even an experienced user will encounter occasional surprises and more sophisticated methods are useful in any event. Errors are convected and diffused across the domain, as discussed in Sect. 3.9, making it essential to achieve as uniform a distribution of truncation error as possible. It is possible, however, to start with a coarse grid and later refine it locally according to an estimate of the discretization error; methods for doing this are called solution adaptive grid methods and will be described in Chap. 11.

Finally, there is the issue of grid generation. When the geometry is complex, this task usually consumes the largest amount of user time by far; it is not unusual for a designer to spend several weeks generating a single grid. Since the accuracy of the solution depends as much (if not more) on the grid quality as on the approximations used for discretization of the equations, grid optimization is a worthwhile investment of time.

Many commercial codes for grid generation exist. Automation of the grid generation process, aimed at reducing the user time and speeding up the process is the major goal in this area. Overlapping grids are easier to generate, but there are geometries in which application of this approach is difficult

due to the existence of too many irregular pieces (e.g. coolant flow in an engine block). Generation of triangular and tetrahedral meshes is easier to automate, which is one of the reasons for their popularity. One usually specifies mesh points on the bounding surface and proceeds from there towards the center of the domain. When a surface grid has been created, tetrahedra that have one base on the surface are generated above it and the process is continued towards the center of the volume along a marching front; the entire process is something like solving an equation by a marching procedure and, indeed, some methods are based on the solution of elliptic or hyperbolic partial differential equations.

Tetrahedral cells are not desirable near walls if the boundary layer needs to be resolved because the first grid point must be very close to the wall while relatively large grid sizes can be used in the directions parallel to the wall. These requirements lead to long thin tetrahedra, creating problems in the approximation of diffusive fluxes. For this reason, some grid generation methods generate first a layer of prisms or hexahedra near solid boundaries, starting with a triangular or quadrilateral discretization of the surface; on top of this layer, a tetrahedral mesh is generated automatically in the remaining part of the domain. An example of such a grid is shown in Fig. 8.2.
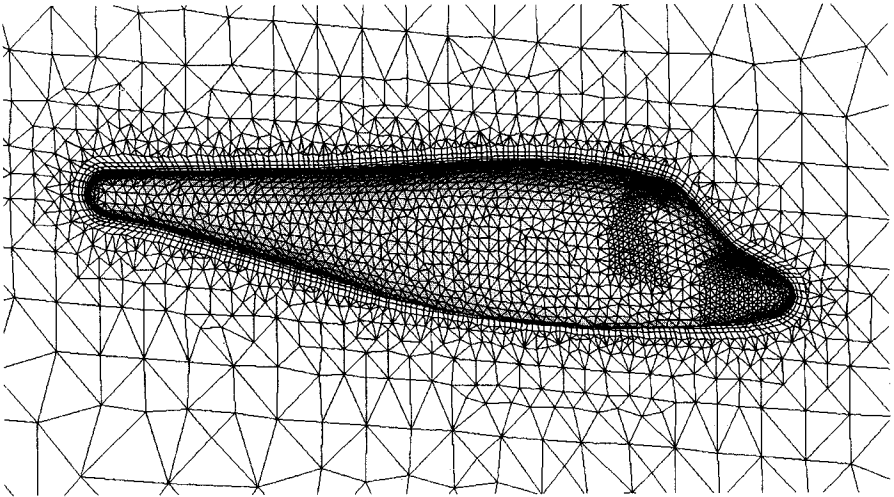
Fig. 8.2. An example of a grid made up of prisms near the walls and tetrahedra in the remaining part of the solution domain (courtesy of ICEM CFD Engineering GmbH; grid generated automatically using ICEM CFD Tetra/Prism grid generator)

This approach enhances grid quality near walls and leads to both more accurate solutions and better convergence of numerical solution methods; however, it can be only used if the solution method allows for mixed control

volume types. In principle, any type of method (FD, FV, FE) can be adapted to this kind of grid.

Another approach to automatic grid generation is to cover the solution domain with a (coarse) Cartesian grid, and adjust the cells cut by domain boundaries to fit the boundary. The problem with this approach is that the cells near boundary are irregular and may require special treatment. However, if this is done on a very coarse level and the grid is then refined several times, the irregularity is limited to a few locations and will not affect the accuracy much but the degree of boundary irregularity is limited.

In order to move the irregular cells further away from walls, one can first create a layer of regular prisms or hexahedra near walls; the outer regular grid is then cut by the surface of the near-wall cell layer. An example of such a grid is shown in Fig. 8.3. This approach allows fast grid generation but requires a solver that can deal with the polyhedral cells created by cutting regular cells with an arbitrary surface. Again, all types of methods can be adapted to this type of grid.
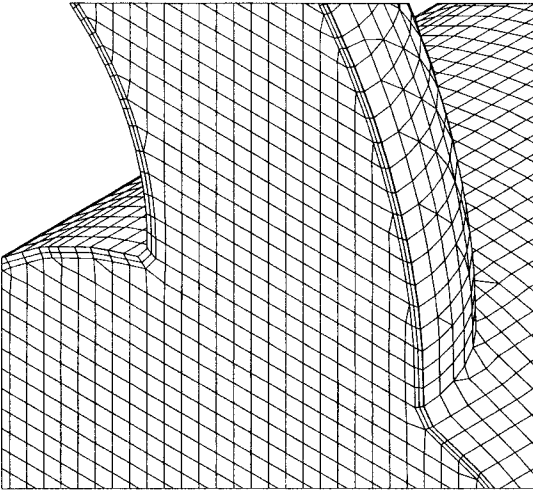
Fig. 8.3. An example of a grid created by combining regular grids near a wall and in the bulk of the solution domain, with irregular cells along surface of the near-wall layer (courtesy of **adapco** Ltd.; grid generated using **samm** grid generator)

If the solution method can be applied on an unstructured grid with cells of varying topology, the grid generation program is subject to few constraints. For example, local grid refinement by subdivision of cells into smaller ones is possible. A non-refined neighbor cell, although it retains its original shape (e.g. a hexahedron), becomes a logical polyhedron, since a face is replaced by a set of sub-faces. The solution domain can first be divided into blocks which can be subdivided into grids with good properties; one has the freedom to

choose the best grid topology (structured H-, O-, or C-grid, or unstructured tetrahedral or hexahedral grid) for each block. The cells on the block interfaces then have faces of irregular shape and have to be treated as polyhedra. An example is shown in Fig. 8.4; the grid contains a block interface on which the faces are irregular. Generation of grids with non-matching interfaces is much simpler than creation of a single-block grid fitted to the whole domain. We again note that the solution method has to allow treatment of polyhedral CVs with an arbitrary numbers of faces; how this can be achieved will be shown below and cell-wise local grid refinement is discussed in Chap. 11.
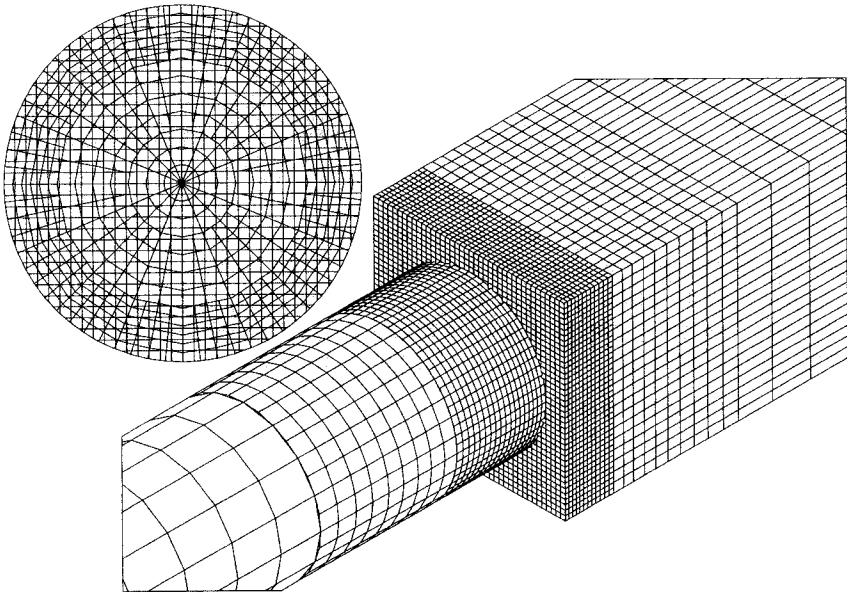
Fig. 8.4. A grid created by combining two blocks with a non-matching interface and cell-wise local refinement (courtesy of ICCM Institute of Computational Continuum Mechanics GmbH; grid generated using the Cometpp grid generator); note the interface with irregular cell faces

If, on the other hand, the solution method requires all CVs to be of a particular type, generation of grids for complex geometries may be difficult. It is not unusual for the process of grid generation to take days or even weeks in some industrial applications of CFD. An efficient grid generator is an essential part of any CFD package.

8.3 The Choice of Velocity Components

In Chap. 1 we discussed issues associated with the choice of components of the momentum. As indicated by Fig. 1.1, only the choice of components in

a fixed basis leads to a fully conservative form of the momentum equations. To ensure momentum conservation, it is desirable to use such a basis and the simplest one is the Cartesian basis. When the flow is three-dimensional, there are no advantages to using any other basis (e.g. grid-oriented, covariant, or contravariant). Only when the choice of another vector basis leads to the simplification of the problem is it worth abandoning the use of Cartesian components. An example of such a case is the flow in a pipe or other axi-symmetric geometries. In the absence of swirl, the velocity vector has only two non-zero components in the polar-cylindrical basis, but three non-zero Cartesian components. The problem therefore has three dependent variables in terms of the Cartesian components, but only two if the polar-cylindrical components are used, a substantial simplification. If the swirl is present, the problem is three-dimensional in any case, so there is no advantage in using polar-cylindrical components rather then Cartesian ones.

8.3.1 Grid-Oriented Velocity Components

If grid-oriented velocity components are used, non-conservative source terms appear in the momentum equations. These account for the redistribution of the momentum between the components. For example, if polar-cylindrical components are used, the divergence of the convection tensor $\rho \mathbf{v}\mathbf{v}$ leads to two such source terms:

- In the momentum equation for the r-component, there is a term $\rho v_\theta^2/r$, which represents the apparent *centrifugal force.* This is not the centrifugal force found in rotating flows (e.g. pump or turbine passages) – it is solely due to the transformation from Cartesian to polar-cylindrical coordinates. This term describes the transfer of θ-momentum into r-momentum due to the change of direction of v_θ.
- In the momentum equation for the θ-component, there is a term $-\rho v_r v_\theta/r$, which represents the apparent *Coriolis force.* This term is source or sink of θ-momentum, depending on the signs of the velocity components.

In general curvilinear coordinates, there are more such source terms (see books by Sedov, 1971; Truesdell, 1977, etc). They involve Christoffel symbols (curvature terms, higher-order coordinate derivatives) and are often a source of numerical errors. The grid is required to be *smooth* – the change of grid direction from point to point must be small. Especially on unstructured grids, in which grid lines are not associated with coordinate directions, this basis is difficult to use.

8.3.2 Cartesian Velocity Components

In this book we shall use Cartesian vector and tensor components exclusively. The discretization and solution techniques remain the same if other

components are used but there are more terms to be approximated. The conservation equations in terms of Cartesian components were given in Chap. 1.

If the FD method is used, one has only to employ the appropriate forms of the divergence and gradient operators for non-orthogonal coordinates (or to transform all derivatives with respect to Cartesian coordinates to the non-orthogonal coordinates). This leads to an increased number of terms, but the conservation properties of the equations remain the same as in Cartesian coordinates, as will be shown below.

In FV methods there is no need for coordinate transformations. When the gradient normal to the CV surface is approximated, one can use a local coordinate transformation, as will be shown below.

8.4 The Choice of Variable Arrangement

In Chap. 7 we mentioned that apart from colocated variable arrangement, various staggered arrangements are possible. While there were no obvious advantages for one or the other for Cartesian grids, the situation changes substantially when non-orthogonal grids are used.

8.4.1 Staggered Arrangements

The staggered arrangement, presented in Chap. 7 for Cartesian grids, is applicable to non-orthogonal grids only if the grid-oriented velocity components are employed. In Fig. 8.5 portions of such a grid are shown, in which the grid lines change direction by 90°. In one case, the contravariant, and in the other case, the Cartesian, velocity components are shown at the staggered locations. Recall that the staggered arrangement was introduced in order to achieve strong coupling between the velocities and the pressure gradient. The goal was to have the velocity component normal to cell face lie between the pressure nodes on either side of that face, see Fig. 7.4. For contravariant or co-variant grid-oriented components, this goal is also achieved on non-orthogonal grids, see Fig. 8.5 (a). For Cartesian components, when the grid lines change direction by 90° a situation like the one shown in Fig. 8.5 (b) arises: the velocity component stored at the cell face makes no contribution to the mass flux through that face, as it is parallel to the face. In order to calculate mass fluxes through such CV faces, one has to use interpolated velocities from surrounding cell faces. This makes the derivation of the pressure-correction equation difficult, and does not ensure the proper coupling of velocities and pressure – oscillations in either may result.

Since, in engineering flows, grid lines often change direction by 180° or more, especially if unstructured grids are used, the staggered arrangement is difficult to use. Some of these problems can be overcome if all Cartesian

components are stored at each CV face. However, this becomes complicated in 3D, especially if CVs of arbitrary shape are allowed. To see how this may be done, interested readers may want to look at the paper by Maliska and Raithby (1984).
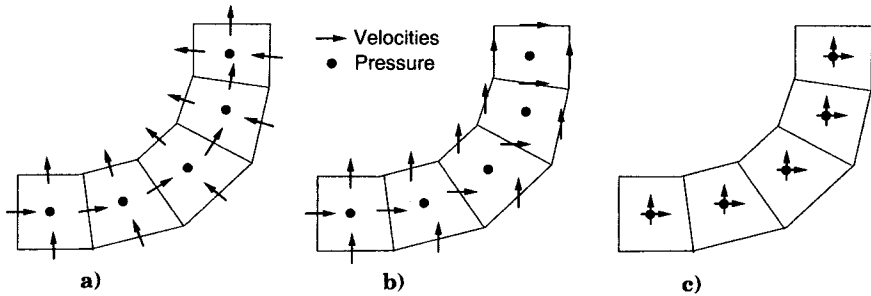
Fig. 8.5. Variable arrangements on a non-orthogonal grid: (a) – staggered arrangement with contravariant velocity components, (b) – staggered arrangement with Cartesian velocity components, (c) – colocated arrangement with Cartesian velocity components

8.4.2 Colocated Arrangement

It was shown in Chap. 7 that the colocated arrangement is the simplest one, since all variables share the same CV, but it requires more interpolation. It is no more complicated than other arrangements when the grid is non-orthogonal, as can be seen from Fig. 8.5 (c). The mass flux through any CV face can be calculated by interpolating the velocities at two nodes on either side of the face; the procedure is the same as on regular Cartesian grids. Most commercial CFD codes use Cartesian velocity components and the colocated arrangement of variables. We shall concentrate on this arrangement.

In what follows we shall describe the new features of the discretization on non-orthogonal grids, building on what has been done in preceding chapters for Cartesian grids.

8.5 Finite Difference Methods

8.5.1 Methods Based on Coordinate Transformation

The FD method is usually used only in conjunction with structured grids, in which case each grid line is a line of constant coordinate ξ_i. The coordinates are defined by the transformation $x_i = x_i(\xi_j), \quad j = 1,2,3$, which is characterized by the Jacobian J:

$$J = \det\left(\frac{\partial x_i}{\partial \xi_j}\right) = \begin{vmatrix} \frac{\partial x_1}{\partial \xi_1} & \frac{\partial x_1}{\partial \xi_2} & \frac{\partial x_1}{\partial \xi_3} \\ \frac{\partial x_2}{\partial \xi_1} & \frac{\partial x_2}{\partial \xi_2} & \frac{\partial x_2}{\partial \xi_3} \\ \frac{\partial x_3}{\partial \xi_1} & \frac{\partial x_3}{\partial \xi_2} & \frac{\partial x_3}{\partial \xi_3} \end{vmatrix} . \tag{8.1}$$

Since we use the Cartesian vector components, we only need to transform the derivatives with respect to Cartesian coordinates into the generalized coordinates:

$$\frac{\partial \phi}{\partial x_i} = \frac{\partial \phi}{\partial \xi_j}\frac{\partial \xi_j}{\partial x_i} = \frac{\partial \phi}{\partial \xi_j}\frac{\beta^{ij}}{J} , \tag{8.2}$$

where β^{ij} represents the cofactor of $\partial x_i/\partial \xi_j$ in the Jacobian J. In 2D this leads to:

$$\frac{\partial \phi}{\partial x} = \frac{1}{J}\left(\frac{\partial \phi}{\partial \xi}\frac{\partial y}{\partial \eta} - \frac{\partial \phi}{\partial \eta}\frac{\partial y}{\partial \xi}\right) . \tag{8.3}$$

The generic conservation equation, which in Cartesian coordinates reads:

$$\frac{\partial(\rho\phi)}{\partial t} + \frac{\partial}{\partial x_j}\left(\rho u_j \phi - \Gamma\frac{\partial \phi}{\partial x_j}\right) = q_\phi , \tag{8.4}$$

transforms to:

$$J\frac{\partial(\rho\phi)}{\partial t} + \frac{\partial}{\partial \xi_j}\left[\rho U_j \phi - \frac{\Gamma}{J}\left(\frac{\partial \phi}{\partial \xi_m}B^{mj}\right)\right] = Jq_\phi , \tag{8.5}$$

where

$$U_j = u_k\beta^{kj} = u_1\beta^{1j} + u_2\beta^{2j} + u_3\beta^{3j} \tag{8.6}$$

is proportional to the velocity component normal to the coordinate surface ξ_j =const. The coefficients B^{mj} are defined as:

$$B^{mj} = \beta^{kj}\beta^{km} = \beta^{1j}\beta^{1m} + \beta^{2j}\beta^{2m} + \beta^{3j}\beta^{3m} . \tag{8.7}$$

The transformed momentum equations contain several additional terms that arise because the diffusive terms in the momentum equations contain a derivative not found in the generic conservation equation, see Eqs. (1.16), (1.18) and (1.19). These terms have the same form as the ones shown above and will not be listed here.

Equation (8.5) has the same form as Eq. (8.4), but each term in the latter is replaced by a sum of three terms in the former. As shown above, these terms contain the first derivatives of the coordinates as coefficients. These are not difficult to evaluate numerically (unlike second derivatives). The unusual feature of non-orthogonal grid is that mixed derivatives appear

in the diffusive terms. In order to show this clearly, we rewrite the Eq. (8.5) in the expanded form:

$$
\begin{aligned}
J\frac{\partial(\rho\phi)}{\partial t} + \frac{\partial}{\partial\xi_1}\left[\rho U_1\phi - \frac{\Gamma}{J}\left(\frac{\partial\phi}{\partial\xi_1}B^{11} + \frac{\partial\phi}{\partial\xi_2}B^{21} + \frac{\partial\phi}{\partial\xi_3}B^{31}\right)\right] + \\
\frac{\partial}{\partial\xi_2}\left[\rho U_2\phi - \frac{\Gamma}{J}\left(\frac{\partial\phi}{\partial\xi_1}B^{12} + \frac{\partial\phi}{\partial\xi_2}B^{22} + \frac{\partial\phi}{\partial\xi_3}B^{32}\right)\right] + \\
\frac{\partial}{\partial\xi_3}\left[\rho U_3\phi - \frac{\Gamma}{J}\left(\frac{\partial\phi}{\partial\xi_1}B^{13} + \frac{\partial\phi}{\partial\xi_2}B^{23} + \frac{\partial\phi}{\partial\xi_3}B^{33}\right)\right] = Jq_\phi \ .
\end{aligned}
\tag{8.8}
$$

All three derivatives of ϕ, which stem from the gradient operator, appear inside each of the outer derivatives, which stem from the divergence operator, see Eq. (1.27). The mixed derivatives of ϕ are multiplied by coefficients B^{mj} with unequal indices, which become zero when the grid is orthogonal, whether it is rectilinear or curvilinear. If the grid is non-orthogonal, their magnitudes relative to the diagonal elements B^{ii} depend on the angles between the grid lines and on the grid aspect ratio. When the angle between grid lines is small and the aspect ratio large, the coefficients multiplying mixed derivatives may be larger than the diagonal coefficients, which can lead to numerical problems (poor convergence, oscillations in the solution etc.). If the non-orthogonality and aspect ratio are moderate, these terms are much smaller than the diagonal ones and cause no problems. The mixed derivative terms are usually treated explicitly, as their inclusion in the implicit computational molecule would make the latter large and solution more expensive. Explicit treatment usually increases the number of outer iterations, but the savings derived from simpler and less expensive inner iterations is far more significant.
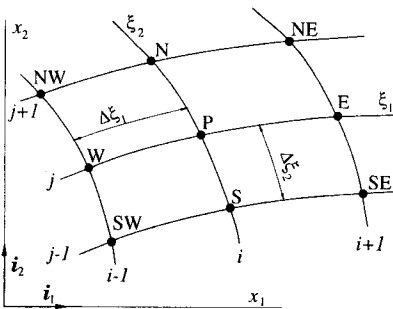
Fig. 8.6. On the coordinate transformation on non-orthogonal grids

The derivatives in Eq. (8.5) can be approximated using one of the FD approaches described in Chap. 3, see Fig. 8.6. The derivatives along curved coordinates are approximated in the same way as those along straight lines.

Coordinate transformations are often presented as a means of converting a complicated non-orthogonal grid into a simple, uniform Cartesian grid (the

spacing in transformed space is arbitrary, but one usually takes $\Delta\xi_i = 1$). Some authors claim that discretization becomes simpler, as the grid in the transformed space appears simpler. This simplification is, however, only apparent: the flow does take place in a complex geometry and this fact cannot be hidden by a clever coordinate transformation. Although the transformed grid does look simpler than the non-transformed one, the information about the complexity is contained in the metric coefficients. While the discretization on the uniform transformed mesh is simple and accurate, calculation of the Jacobian and other geometric information is not trivial and introduces additional discretization errors; i.e. it is here that the real difficulty has been hidden.

The mesh spacing $\Delta\xi_i$ need not be specified explicitly. The volume in physical space, $\Delta\Omega$, is defined as:

$$\Delta\Omega = J\Delta\xi_1\Delta\xi_2\Delta\xi_3 \; . \tag{8.9}$$

If we multiply the whole equation by $\Delta\xi_1\Delta\xi_2\Delta\xi_3$, and replace $J\Delta\xi_1\Delta\xi_2\Delta\xi_3$ everywhere by $\Delta\Omega$, then the mesh spacings $\Delta\xi_i$ disappear in all terms. If central differences are used to approximate the coefficients β^{ij}, e.g. in 2D (see Fig. 8.6):

$$\begin{aligned} \beta^{11} &= \frac{\partial y}{\partial \eta} \approx \frac{y_\mathrm{N} - y_\mathrm{S}}{2\,\Delta\eta} \; , \\ \beta^{12} &= -\frac{\partial x}{\partial \eta} \approx \frac{x_\mathrm{N} - x_\mathrm{S}}{2\,\Delta\eta} \; , \end{aligned} \tag{8.10}$$

the final discretized terms will involve only the differences in Cartesian coordinates between neighbor nodes and the volumes of imaginary cells around each node. Therefore, all we need is to construct such non-overlapping cells around each grid node and calculate their volume – the coordinates ξ_i need not be assigned any value and the coordinate transformation is hidden.

8.5.2 Method Based on Shape Functions

Although nobody seems to have tried it, the FD method can be applied to arbitrary unstructured grids. One would have to prescribe a differentiable shape function (probably a polynomial) which describes the variation of the variable ϕ in the vicinity of a particular grid point. The coefficients of the polynomial would be obtained by fitting the shape function to the values of ϕ at a number of surrounding nodes. There would be no need to transform any term in the equation, as the shape function could be differentiated analytically to provide expressions for the first and second derivatives with respect to Cartesian coordinates at the central grid point in terms of the variable values at surrounding nodes and geometrical parameters. The resulting coefficient matrix would be sparse but it would not have a diagonal structure unless the grid is structured.

One can also allow different shape functions to be used, depending on the local grid topology. This would lead to a different number of neighbors in computational molecules, but a solver that can deal with this complexity can easily be devised (e.g. conjugate-gradient type solvers).

One can devise a finite difference method that does not need a grid at all; a set of discrete points adequately distributed over the solution domain is all that is needed. One would then locate a certain number of near neighbors of each point to which one could fit a suitable shape function; the shape function could then be differentiated to obtain approximations of the derivatives at that point. Such a method cannot be fully conservative, but this is not a problem if the points are sufficiently densely spaced.

It appears easier to distribute points in space than to create suitable control volumes or elements of good quality. For example, one could first place points on the surface, then add points a short distance away in the direction normal to the surface. A second set of points could be regularly distributed in the solution domain, with higher density near boundaries. Then the two sets of points can be checked for overlap, and, where points are too close to each other, they can be deleted or moved. Local refinement is very easy, one needs merely insert more points between the existing ones.

The only tricky thing would be the derivation of a suitable pressure or pressure-correction equation; however, this could be achieved following the methods presented in the following sections. We hope to see methods of this kind in future editions of this work.

The principles described above apply to all equations. The special features of deriving the pressure or pressure-correction equation or implementing the boundary conditions in FD methods on non-orthogonal grids will not be dealt with here in detail, as the extension of techniques given so far is straightforward.

8.6 Finite Volume Methods

The FV method starts from the conservation equation in integral form, e.g. the generic conservation equation:

$$\frac{\partial}{\partial t}\int_{\Omega}\rho\phi\,\mathrm{d}\Omega + \int_{S}\rho\phi\boldsymbol{v}\cdot\boldsymbol{n}\,\mathrm{d}S = \int_{S}\Gamma\mathrm{grad}\phi\cdot\boldsymbol{n}\,\mathrm{d}S + \int_{\Omega}q_{\phi}\,\mathrm{d}\Omega\,. \quad (8.11)$$

The principles of FV discretization were described in Chap. 4. These are independent of the type of grid used; however, there are several new features that non-orthogonal or arbitrary unstructured grids bring with them; these will be considered in the following sections.

8.6.1 Approximation of Convective Fluxes

We shall use the midpoint rule approximation of the surface and volume integrals exclusively. We look first at the calculation of mass fluxes. Only the east side of a 2D CV shown in Fig. 8.7 will be considered; the same approach applies to other faces – only the indices need be changed. The CV may have any number of faces; the analysis is not restricted to quadrilateral CVs like the one shown in Fig. 8.7.

The midpoint rule approximation of the mass flux leads to:

$$\dot{m}_{\mathrm{e}} = \int_{S_{\mathrm{e}}} \rho \boldsymbol{v} \cdot \boldsymbol{n} \, \mathrm{d}S \approx (\rho \boldsymbol{v} \cdot \boldsymbol{n})_{\mathrm{e}} S_{\mathrm{e}} \; . \tag{8.12}$$

The unit normal vector at the face "e" is defined by:

$$\boldsymbol{n}_{\mathrm{e}} S_{\mathrm{e}} = S_{\mathrm{e}}^{i} \boldsymbol{i}_{i} = (y_{\mathrm{ne}} - y_{\mathrm{se}}) \, \boldsymbol{i} - (x_{\mathrm{ne}} - x_{\mathrm{se}}) \, \boldsymbol{j} \; , \tag{8.13}$$

and the surface area, S_{e}, is:

$$S_{\mathrm{e}} = \sqrt{(S_{\mathrm{e}}^{x})^2 + (S_{\mathrm{e}}^{y})^2} \; . \tag{8.14}$$

With these definitions the expression for the mass flux becomes:

$$\dot{m}_{\mathrm{e}} = \rho_{\mathrm{e}} \left(S^{x} u_{x} + S^{y} u_{y} \right)_{\mathrm{e}} \; . \tag{8.15}$$

The difference between a Cartesian and a non-orthogonal grid is that, in the latter case, the surface vector has components in more than one Cartesian direction and all the velocity components contribute to the mass flux. Each Cartesian velocity component is multiplied by the corresponding surface vector component (projection of the cell face onto a Cartesian coordinate plane), see Eq. (8.15).
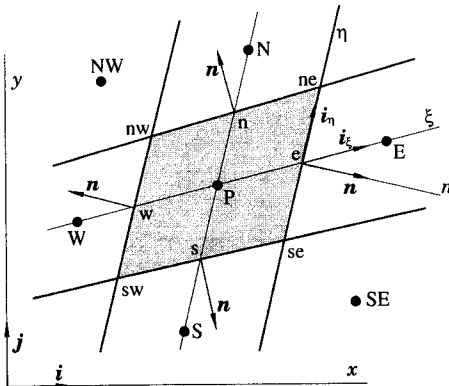
Fig. 8.7. A typical 2D control volume and the notation used

The convective flux of any transported quantity is usually calculated by assuming that the mass flux is known which, with the midpoint rule approximation, leads to:

$$F_{\mathrm{e}}^{\mathrm{c}} = \int_{S_{\mathrm{e}}} \rho \phi \boldsymbol{v} \cdot \boldsymbol{n} \, \mathrm{d}S \approx \dot{m}_{\mathrm{e}} \phi_{\mathrm{e}} \, , \tag{8.16}$$

where ϕ_{e} is the value of ϕ at the center of the cell face. The simplest approximation of second order is obtained by linear interpolation between the two nodes on either side of the face. Other approximations, some of which were described in Chap. 4 for Cartesian grids, can be used. The interpolation is usually performed by treating the piecewise linear lines as if they were straight; if the line changes direction at the cell face, an additional error is introduced. Another possibility is to fit the variation of ϕ in the vicinity of the face to a polynomial.

On structured non-orthogonal grids one can use higher-order integration and interpolation techniques to approximate convective fluxes, as described in Chap. 4 for a 2D case. However, if the grid is unstructured and involves CVs of arbitrary numbers of faces, use of linear interpolation and the midpoint rule approximation seem to offer the best compromise among accuracy, generality, and simplicity. Indeed, a computer code which uses these techniques is simple, even for CVs of arbitrary shape. This technique also facilitates use of local grid refinement, described in Chap. 11, which can be used to achieve high accuracy at a lower cost than through use of higher-order techniques.

8.6.2 Approximation of Diffusive Fluxes

The midpoint rule applied to the integrated diffusive flux gives:

$$F_{\mathrm{e}}^{\mathrm{d}} = \int_{S_{\mathrm{e}}} \Gamma \, \mathrm{grad} \, \phi \cdot \boldsymbol{n} \, \mathrm{d}S \approx (\Gamma \, \mathrm{grad} \, \phi \cdot \boldsymbol{n})_{\mathrm{e}} S_{\mathrm{e}} \, . \tag{8.17}$$

The gradient of ϕ at the cell face center can be expressed either in terms of the derivatives with respect to global Cartesian coordinates or local orthogonal coordinates (n, t), e.g. in 2D:

$$\mathrm{grad} \, \phi = \frac{\partial \phi}{\partial x} \boldsymbol{i} + \frac{\partial \phi}{\partial y} \boldsymbol{j} = \frac{\partial \phi}{\partial n} \boldsymbol{n} + \frac{\partial \phi}{\partial t} \boldsymbol{t} \, , \tag{8.18}$$

where n and t represent the coordinate directions normal and tangential to the surface, respectively (in 3D there is a third coordinate s, which is orthogonal to both n and t, and tangential to the surface).

There are many ways to approximate the derivative normal to the cell face or the gradient vector at the cell center; we shall describe only few of them. If the variation of ϕ in the vicinity of the cell face is described by a shape function, it is then possible to differentiate this function at the 'e' location to find the derivatives with respect to the Cartesian coordinates. The diffusive flux is then:

$$F_{\mathrm{e}}^{\mathrm{d}} = \Gamma_{\mathrm{e}} \sum_i \left(\frac{\partial \phi}{\partial x_i} \right)_{\mathrm{e}} S_{\mathrm{e}}^i \, . \tag{8.19}$$

This is easy to implement *explicitly;* an implicit version may be complicated, depending on the order of the shape function and the number of nodes involved.

Another way to calculate derivatives at the cell face is to obtain them first at CV centers, and then interpolate them to the cell faces in the way ϕ_e was. A simple way of doing this is provided by the Gauss' theorem; we approximate the derivative at the CV center by the average value over the cell:

$$\left(\frac{\partial \phi}{\partial x_i}\right)_\mathrm{P} \approx \frac{\int_\Omega \frac{\partial \phi}{\partial x_i}\,\mathrm{d}\Omega}{\Delta\Omega} \ . \tag{8.20}$$

Then we can consider the derivative $\partial\phi/\partial x_i$ as the divergence of the vector $\phi \boldsymbol{i}_i$ and transform the volume integral in the above equation using Gauss' theorem into a surface integral:

$$\int_\Omega \frac{\partial \phi}{\partial x_i}\,\mathrm{d}\Omega = \int_S \phi\boldsymbol{i}_i \cdot \boldsymbol{n}\,\mathrm{d}S \approx \sum_c \phi_c S_c^i \ , \quad c = \mathrm{e,n,w,s,}\ldots \tag{8.21}$$

This shows that one can calculate the gradient of ϕ with respect to x at the CV center by summing the products of ϕ with the x-components of the surface vectors at all faces of the CV and dividing the sum by the CV volume:

$$\left(\frac{\partial \phi}{\partial x_i}\right)_\mathrm{P} \approx \frac{\sum_c \phi_c S_c^i}{\Delta\Omega} \ . \tag{8.22}$$

For ϕ_c we can use the values used to calculate the convective fluxes, although one need not necessarily use the same approximation for both terms. For Cartesian grids and linear interpolation, the standard central difference approximation is obtained:

$$\left(\frac{\partial \phi}{\partial x_i}\right)_\mathrm{P} \approx \frac{\phi_\mathrm{E} - \phi_\mathrm{W}}{2\,\Delta x} \ . \tag{8.23}$$

Cell-center gradients can also be approximated within second order by using linear shape functions; if we assume linear variation of ϕ between two neighbor cell centers, e.g. P and E, we may write:

$$\phi_\mathrm{E} - \phi_\mathrm{P} = (\mathrm{grad}\,\phi)_\mathrm{P} \cdot (\boldsymbol{r}_\mathrm{E} - \boldsymbol{r}_\mathrm{P}) \ . \tag{8.24}$$

We can write as many such equations as there are neighbors for the cell around node P; however, we need to compute only three derivatives $\partial\phi/\partial x_i$. With the help of least-squares methods, the derivatives can be explicitly computed for arbitrary CV shapes.

The derivatives calculated in this way can be interpolated to the cell face and the diffusive flux can be calculated from Eq. (8.19). The problem with

this approach is that an oscillatory solution may be generated in the course of the iteration procedure and the oscillations will not be detected. How this can be avoided is described below.

For explicit methods, this approach is very simple and effective. It is, however, not suitable for implementation in an implicit method since it produces large computational molecules. The *deferred correction* approach described in Sect. 5.6 offers a way around this problem and helps to eliminate the oscillations. It consists of using a simple approximation to the diffusive flux implicitly and creating a right hand side which is the difference between the correct and approximate fluxes. With good choices of approximations the convergence of the implicit method is not impaired by the deferred correction.

A good approximation for the implicit part of the method is easily found. If we use the local (n, t, s) orthogonal coordinate system attached to the cell face center, then only the derivative in the n-direction contributes to the diffusive flux:

$$F_{\mathrm{e}}^{\mathrm{d}} = \Gamma_{\mathrm{e}} \left(\frac{\partial \phi}{\partial n} \right)_{\mathrm{e}} S_{\mathrm{e}} \; . \tag{8.25}$$

On a Cartesian grid, $n = x$ at the "e" face and we can use the central difference approximation:

$$\left(\frac{\partial \phi}{\partial n} \right)_{\mathrm{e}} \approx \frac{\phi_{\mathrm{E}} - \phi_{\mathrm{P}}}{L_{\mathrm{P,E}}} \; , \tag{8.26}$$

where $L_{\mathrm{P,E}}$ is the distance between nodes E and P, $|\boldsymbol{r}_{\mathrm{E}} - \boldsymbol{r}_{\mathrm{P}}|$ ($L_{\mathrm{P,E}} = \Delta x$ on a uniform Cartesian grid). The interpolated cell center gradient gives (on a uniform Cartesian grid):

$$\overline{\left(\frac{\partial \phi}{\partial n} \right)}_{\mathrm{e}} = \frac{1}{2} \, \frac{\phi_{\mathrm{E}} - \phi_{\mathrm{W}}}{2\,\Delta x} + \frac{1}{2} \, \frac{\phi_{\mathrm{EE}} - \phi_{\mathrm{P}}}{2\,\Delta x} \; . \tag{8.27}$$

An oscillatory distribution of ϕ in x-direction shown in Fig. 8.8 will not contribute to this gradient, since both $\phi_{\mathrm{E}} - \phi_{\mathrm{W}}$ and $\phi_{\mathrm{EE}} - \phi_{\mathrm{P}}$ are zero and so are the gradients at each cell center. However, the gradients are large at cell faces. Oscillations do indeed develop during the iteration process. The obvious deferred correction approach:

$$F_{\mathrm{e}}^{\mathrm{d}} = F_{\mathrm{e}}^{\mathrm{d,impl}} + \left[F_{\mathrm{e}}^{\mathrm{d,expl}} - F_{\mathrm{e}}^{\mathrm{d,impl}} \right]^{\mathrm{old}} \; , \tag{8.28}$$

in which 'impl' and 'expl' denote the implicit (using Eq. (8.26)) and the explicit (using Eq. (8.27)) flux approximation and 'old' means value from previous iteration, allows oscillatory solutions to develop. A similar problem appears in the derivation of the pressure-correction equation for colocated arrangements and was discussed in Chap. 7.
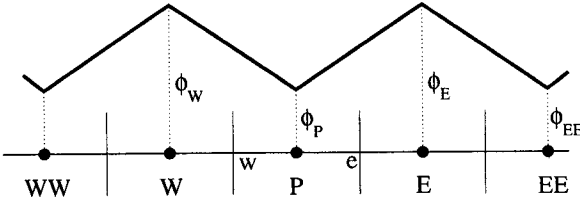
Fig. 8.8. On the approximation of gradients at cell faces and avoidance of oscillatory solutions

Muzaferija (1994) recognized the problem and suggested an effective cure. He noted that, when the line connecting nodes P and E is nearly orthogonal to the cell face, the derivative with respect to n can be approximated by a derivative with respect to the coordinate, ξ, along that line. He suggested to use as an implicit flux approximation the expression (see Eq. (8.25)):

$$F_{\mathrm{e}}^{\mathrm{d}} = \Gamma_{\mathrm{e}} S_{\mathrm{e}} \left(\frac{\partial \phi}{\partial \xi} \right)_{\mathrm{e}} = \Gamma_{\mathrm{e}} S_{\mathrm{e}} \frac{\phi_E - \phi_P}{L_{\mathrm{P,E}}} \ . \tag{8.29}$$

If the line connecting nodes P and E is orthogonal to the cell face, this is a second order accurate approximation and the deferred correction term should be zero. When the grid is non-orthogonal, the deferred correction term must contain the difference between the gradients in the ξ and n directions. The deferred correction formula suggested by Muzaferija (1994) is:

$$F_{\mathrm{e}}^{\mathrm{d}} = \Gamma_{\mathrm{e}} S_{\mathrm{e}} \left(\frac{\partial \phi}{\partial \xi} \right)_{\mathrm{e}} + \Gamma_{\mathrm{e}} S_{\mathrm{e}} \left[\overline{\left(\frac{\partial \phi}{\partial n} \right)_{\mathrm{e}}} - \overline{\left(\frac{\partial \phi}{\partial \xi} \right)_{\mathrm{e}}} \right]^{\mathrm{old}} \ . \tag{8.30}$$

The first term on the right hand side is treated implicitly while the second term is the deferred correction. The deferred correction term is calculated using interpolated cell center gradients (e.g. obtained using Gauss' theorem) in n and ξ directions, i.e.:

$$\overline{\left(\frac{\partial \phi}{\partial n} \right)_{\mathrm{e}}} = \overline{(\mathrm{grad}\, \phi)_{\mathrm{e}}} \cdot \boldsymbol{n} \ ; \quad \overline{\left(\frac{\partial \phi}{\partial \xi} \right)_{\mathrm{e}}} = \overline{(\mathrm{grad}\, \phi)_{\mathrm{e}}} \cdot \boldsymbol{i}_\xi \ , \tag{8.31}$$

where $\boldsymbol{i}_\xi$ is the unit vector in the ξ-direction. The final expression for the approximation to the diffusive flux through the cell face 'e' can now be written:

$$F_{\mathrm{e}}^{\mathrm{d}} = \Gamma_{\mathrm{e}} S_{\mathrm{e}} \frac{\phi_E - \phi_P}{L_{\mathrm{P,E}}} + \Gamma_{\mathrm{e}} S_{\mathrm{e}} \overline{(\mathrm{grad}\, \phi)_{\mathrm{e}}}^{\mathrm{old}} \cdot (\boldsymbol{n} - \boldsymbol{i}_\xi) \ , \tag{8.32}$$

i.e. CDS is used to approximate the derivative in ξ direction. Thus, the deferred correction term, labeled "old", becomes zero when $\boldsymbol{i}_\xi = \boldsymbol{n}$, as required. When the non-orthogonality is not severe, this term is small compared to the implicit term and the convergence rate of the implicit solution method is not impaired substantially.

The non-orthogonality of the lines defining the CV boundaries is not relevant – only the angle between the cell face normal n and the line ξ connecting the cell centers on either side is important. A 2D grid of equilateral triangles is orthogonal in the above sense, since the directions of ξ and n coincide. It is also uniform in the sense that the distances from cell face centers to CV centers are equal. It is much easier to optimize a grid with respect to the angle between ξ and n than the angles at the CV corners, especially if CVs of different topology are used in the same grid.

This diffusive flux approximation (8.32) prevents oscillatory solutions. It is very simple to implement, since only the cell face surface vector and the positions of CV centers are needed. It is of second order accuracy on uniform grids, and when the grid is refined systematically, the convergence behavior is of second order even when the grid is non-uniform (see Sect. 3.3.1 for details). It is applicable to CVs of arbitrary shape and can be adapted to schemes of higher order.

The above scheme makes the calculation of derivatives with respect to Cartesian coordinates very simple. By using expressions (8.22) and (8.31), one can calculate the derivative in any direction. A subroutine which calculates derivatives is easy to program and can be used for all variables. There is no need to transform the equations from Cartesian coordinates into another system. This is especially handy when implementation of turbulence models is considered, see Chap. 9 (especially for the complicated ones): the model equations are usually complicated enough in Cartesian coordinates – transformation to non-orthogonal coordinates makes them even more complex. It is also very easy to validate the part of the computer code that calculates gradients, by setting ϕ to be an analytic function whose derivatives can be computed exactly.

When structured grids are used, one can also start from the expression (8.25) and transform the derivative with respect to n into derivatives with respect to local coordinates (ξ, η, ζ), where ξ connects the CV centers on either side and the other two coordinates lie in the face and follow the grid directions. This procedure is similar to the one described in the preceding section and will not be discussed in detail here. A 2D code that uses non-orthogonal structured grids is available via Internet; see appendix for details.

In the momentum equations, the diffusive flux contains a few more terms than does the corresponding term in the generic conservation equation, e.g. for u_i:

$$F_{\mathrm{e}}^{\mathrm{d}} = \int_{S_{\mathrm{e}}} \mu \, \mathrm{grad}\, u_i \cdot \boldsymbol{n} \, \mathrm{d}S + \underline{\int_{S_{\mathrm{e}}} \mu \frac{\partial u_j}{\partial x_i} \boldsymbol{i}_j \cdot \boldsymbol{n} \, \mathrm{d}S} \, . \tag{8.33}$$

The underlined term is absent in the generic conservation equation. If ρ and μ are constant, the sum of underlined terms over all CV faces is zero by virtue of the continuity equation, see Sect. 7.1. If ρ and μ are not constant, they – except near shocks – vary smoothly and the integral of underlined terms over

the whole CV surface is smaller than the integral of the principal term. For this reason, the underlined term is usually treated explicitly. As shown above, the derivatives are easily calculated at the cell face using the derivatives at the CV center.

In the above approximations it was assumed that the line connecting nodes P and E passes through the cell face center 'e'. In that case the approximation of the surface integral is second order accurate (midpoint rule). When the grid is irregular, the line connecting P and E may not pass through the cell face center, and the approximations for ϕ_{e} and $(\partial\phi/\partial\xi)_{\mathrm{e}}$ used above are second order accurate at a point 'e′', see Fig. 8.9. The approximation to the surface integral is no longer second order accurate but the additional error is small if 'e′' is not far from 'e'. If 'e′' is close to the corners 'se' or 'ne', the approximation becomes first order accurate.
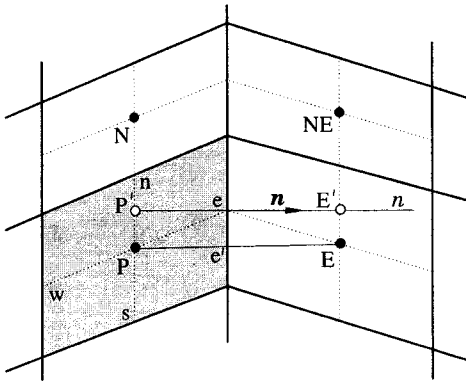
Fig. 8.9. An alternative way of calculating cell face values and gradients

The second order accuracy of the midpoint rule integral approximation can be preserved on irregular grids if the values of the variable and its gradient are calculated at the cell face center with second order approximations. These can be obtained by using appropriate shape functions. Alternatively, one can use values at auxiliary nodes P′ and E′, which lie at the intersection of the cell face normal n and straight lines connecting nodes P and N or E and NE, respectively, see Fig. 8.9. Both the cell face value of the variable and its gradient can then be approximated as on a Cartesian grid from the values at P′ and E′. In order to avoid extended computational molecules in implicit methods, the deferred correction approach can be used: the implicit terms are based on the values at nodes P and E ignoring grid irregularity (i.e. using values at 'e′'), while the difference between the implicit term and the more accurate approximation is treated explicitly. For example, the diffusive flux approximation can be implemented in the following way, using a modified version of expression (8.30):

$$F_{\mathrm{e}}^{\mathrm{d}} = \Gamma_{\mathrm{e}} S_{\mathrm{e}} \left(\frac{\partial \phi}{\partial \xi}\right)_{\mathrm{e}'} + \Gamma_{\mathrm{e}} S_{\mathrm{e}} \left[\overline{\left(\frac{\partial \phi}{\partial n}\right)_{\mathrm{e}}} - \overline{\left(\frac{\partial \phi}{\partial \xi}\right)_{\mathrm{e}'}}\right]^{\mathrm{old}} . \tag{8.34}$$

The normal gradient at a cell face center can be calculated using the usual central difference approximation:

$$\left(\frac{\partial \phi}{\partial n}\right)_{\mathrm{e}} \approx \frac{\phi_{\mathrm{E}'} - \phi_{\mathrm{P}'}}{L_{\mathrm{P}',\mathrm{E}'}} , \tag{8.35}$$

where $L_{\mathrm{P}',\mathrm{E}'}$ stands for the distance between P′ and E′, i.e. $L_{\mathrm{P}',\mathrm{E}'} = |\boldsymbol{r}_{\mathrm{E}'} - \boldsymbol{r}_{\mathrm{P}'}|$. The values $\phi_{\mathrm{E}'}$ and $\phi_{\mathrm{P}'}$ can be calculated either by using bilinear interpolation (which is suitable on structured grids) or by using the gradient at CV center (suitable for arbitrary CV shape):

$$\phi_{\mathrm{P}'} = \phi_{\mathrm{P}} + (\mathrm{grad}\, \phi)_{\mathrm{P}} \cdot (\boldsymbol{r}_{\mathrm{P}'} - \boldsymbol{r}_{\mathrm{P}}) . \tag{8.36}$$

The above scheme is the simplest one that is second order accurate. Higher-order methods must use shape functions, producing a kind of blended FE/FV method. An example of this approach is given in Sect. 8.7.

8.6.3 Approximation of Source Terms

The midpoint rule approximates a volume integral by the product of the CV center value of the integrand and the CV volume:

$$Q_{\mathrm{P}}^{\phi} = \int_{\Omega} q_{\phi}\, \mathrm{d}\Omega \approx q_{\phi,\mathrm{P}}\, \Delta\Omega . \tag{8.37}$$

This approximation is independent of the CV shape and is approximately of second order accuracy.

The calculation of the cell volume deserves some attention. If the grid is structured, simple formulae are available; for example, for 2D quadrilaterals we can use the vector product of the two diagonals:

$$\begin{aligned} \Delta\Omega = & \frac{1}{2} [(\boldsymbol{r}_{\mathrm{ne}} - \boldsymbol{r}_{\mathrm{sw}}) \times (\boldsymbol{r}_{\mathrm{nw}} - \boldsymbol{r}_{\mathrm{se}})] = \\ & \frac{1}{2} [(x_{\mathrm{ne}} - x_{\mathrm{sw}})(y_{\mathrm{nw}} - y_{\mathrm{se}}) - (y_{\mathrm{ne}} - y_{\mathrm{sw}})(x_{\mathrm{nw}} - x_{\mathrm{se}})] , \end{aligned} \tag{8.38}$$

where $\boldsymbol{r}_{\mathrm{ne}}$ is the position vector of the point 'ne', see Fig. 8.7. Suitable expressions for arbitrary 2D and 3D CVs will be given below.

Let us look now at the pressure terms in the momentum equations. They can be treated either as conservative forces on the CV surface, or as non-conservative body forces. In the first case we have (in the 2D equation for u_x, using the midpoint rule approximation):

$$Q_{\mathrm{P}}^{p} = -\int_{S} p\,\boldsymbol{i}\cdot\boldsymbol{n}\,\mathrm{d}S \approx \sum_{c} p_c S_c^x = \tag{8.39}$$

$$-p_{\mathrm{e}}(y_{\mathrm{ne}} - y_{\mathrm{se}}) + p_{\mathrm{w}}(y_{\mathrm{nw}} - y_{\mathrm{sw}}) + p_{\mathrm{n}}(y_{\mathrm{ne}} - y_{\mathrm{nw}}) - p_{\mathrm{s}}(y_{\mathrm{se}} - y_{\mathrm{sw}}) \,.$$

In the second case we get:

$$Q_{\mathrm{P}}^{p} = -\int_{\Omega} \frac{\partial p}{\partial x}\,\mathrm{d}\Omega \approx -\left(\frac{\partial p}{\partial x}\right)_{\mathrm{P}} \Delta\Omega \,. \tag{8.40}$$

The first approach is fully conservative. The second is conservative (and equivalent to the first one) if the derivative $\partial p/\partial x$ is calculated using Gauss' theorem. If the pressure gradient with respect to x is transformed into gradients with respect to ξ and η for a local coordinate system at the CV center, one obtains:

$$Q_{\mathrm{P}}^{p} \approx -(p_{\mathrm{e}} - p_{\mathrm{w}})(y_{\mathrm{n}} - y_{\mathrm{s}}) + (p_{\mathrm{n}} - p_{\mathrm{s}})(y_{\mathrm{e}} - y_{\mathrm{w}}) \,. \tag{8.41}$$

One can calculate the derivative at CV center by differentiating a shape function. These approaches are in general not conservative.

Treatment of pressure terms in the equations for u_y (and in 3D cases u_z) are similar to those given above for u_x.

8.6.4 Three-Dimensional Grids

In 3D, the cell faces are not necessarily planar. To calculate cell volumes and cell face surface vectors, suitable approximations are necessary. A simple method is to represent the cell face by a set of plane triangles. For the hexahedra used in structured grids, Kordula and Vinokur (1983) suggested decomposing each CV into eight tetrahedra (each CV face being subdivided into two triangles) so that no overlapping occurs.

Another way to calculate cell volumes for arbitrary CVs is based on Gauss theorem. By using the identity $1 = \mathrm{div}(x\boldsymbol{i})$, one can calculate the volume as:

$$\Delta\Omega = \int_{\Omega} \mathrm{d}\Omega = \int_{\Omega} \mathrm{div}(x\boldsymbol{i})\,\mathrm{d}\Omega = \int_{S} x\boldsymbol{i}\cdot\boldsymbol{n}\,\mathrm{d}S \approx \sum_{c} x_c\, S_c^x \,, \tag{8.42}$$

where 'c' denotes cell faces and S_c^x is the x-component of the cell face surface vector (see Fig. 8.10):

$$\boldsymbol{S}_c = S_c\,\boldsymbol{n} = S_c^x\,\boldsymbol{i} + S_c^y\,\boldsymbol{j} + S_c^z\,\boldsymbol{k} \,. \tag{8.43}$$

Instead of $x\boldsymbol{i}$, one can also use $y\boldsymbol{j}$ or $z\boldsymbol{k}$, in which case one has to sum the products of $y_c\,S_c^y$ or $z_c\,S_c^z$. If each cell face is defined in the same way for both CVs to which it is common, the procedure ensures that no overlapping occurs and that the sum of all CV volumes equals the volume of the solution domain.

An important issue is the definition of the surface vectors at the cell faces. The simplest approach is to decompose them into triangles with one common vertex, see Fig. 8.10. The areas and surface normal vectors of triangles are easily computed. The surface normal vector for the whole cell face is then the sum of surface vectors of all the triangles (see face 'c$_1$' in Fig. 8.10):

$$\boldsymbol{S}_c = \frac{1}{2}\sum_{i=3}^{N_\mathrm{v}} [(\boldsymbol{r}_{i-1} - \boldsymbol{r}_1) \times (\boldsymbol{r}_i - \boldsymbol{r}_1)] \ , \tag{8.44}$$

where N_v is the number of vertices in the cell face and $\boldsymbol{r}_i$ is the position vector of the vertex i. Note that there are $N_\mathrm{v} - 2$ triangles. The above expression is correct even if the cell face is twisted or convex. The choice of the common vertex is not important.
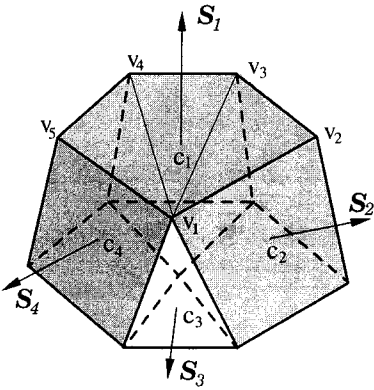
Fig. 8.10. On the calculation of cell volume and surface vectors for arbitrary control volumes

The cell face center can be found by averaging the coordinates of the center of each triangle (which is itself the average of its vertex coordinates) weighted by its area. The area of the cell face is approximated by the magnitude of its surface vector, e.g.:

$$S_\mathrm{e} = |\boldsymbol{S}_\mathrm{e}| = \sqrt{(S_\mathrm{e}^x)^2 + (S_\mathrm{e}^y)^2 + (S_\mathrm{e}^z)^2} \ . \tag{8.45}$$

Note that only projections of the cell face onto Cartesian coordinate planes are required. These are exact when the CV edges are straight, as they are assumed to be.

Further details of discretization on 3D grids will not be presented; the techniques described for 2D problems are easily extended to 3D, and the increased complexity is of strictly geometrical nature.

It is worth noting that the derivation of high order FV methods is more difficult than construction of FD methods of high order. In FD methods, we only have to approximate the derivatives at a grid point with higher order

approximations, which is relatively easy to do on structured grids (see Chap. 3). In FV methods, there are two approximation levels: approximation of the integrals and approximation of the variable values at locations other than CV center. The second order accuracy of the midpoint rule and linear interpolation is the highest accuracy achievable with single-point approximations. Any higher order FV scheme requires interpolation of higher order at more than one cell face location. This is manageable on structured grids, but rather difficult on unstructured grids. For the sake of simplicity of implementation, extension, debugging and maintenance, second order accuracy appears to be the best compromise. Only when very high accuracy is required (discretization errors below 1%) do higher order methods become cost-effective. One also has to bear in mind that higher order methods produce more accurate results than a second order method only if the grid is *sufficiently fine.* If the grid is not fine enough, higher order methods may produce oscillatory solutions, and the average error may be higher than for a second order scheme. Higher order schemes also require more memory and computing time per grid point than second order schemes. For industrial applications, for which errors of the order of 1% are acceptable, a second order scheme coupled with local grid refinement offers the best combination of accuracy, simplicity of programming and code maintenance, robustness, and efficiency.

8.6.5 Block-Structured Grids

Structured grids are difficult, sometimes impossible, to construct for complex geometries. For example, to compute the flow around a circular cylinder in a free stream, one can easily generate a structured O-type grid around it, but if the cylinder is located in a narrow duct, this is no longer possible; see Fig. 2.2. In such a case, block-structured grids provide a useful compromise between the simplicity and wide variety of solvers available for structured grids and ability to handle complex geometries that unstructured grids allow. The idea is to use a regular data structure (lexicographic ordering) within while constructing the blocks so as to fill the irregular domain. Many approaches are possible. Some use overlapping blocks (e.g. Hinatsu and Ferziger, 1991; Perng and Street, 1991; Zang and Street, 1995; Hubbard and Chen, 1994, 1995). Others rely on non-overlapping blocks (e.g. Coelho et al., 1991; Lilek et al., 1997b). We shall describe one approach that used non-overlapping blocks. It is also well suited for use on parallel computers (see Chap. 11); normally, the computing for each block is assigned to a separate processor.

The solution domain is first subdivided into several sub-domains in such a way that each sub-domain can be fitted with a structured grid with good properties (not too non-orthogonal, individual CV aspect ratios not too large). An example is shown in Fig. 2.2. Within each block the indices i and j are used to identify the CVs, but we also need a block-identifier. The data is stored in a one dimensional array. The index of the node (i, j) in block 3 within that one-dimensional array is (see Table 3.2):

$$l = O_3 + (i-1)N_j^3 + j \ ,$$

where O_3 is the offset for block 3 (the number of nodes in all preceding blocks, i.e. $N_i^1 N_j^1 + N_i^2 N_j^2$) and N_i^m and N_j^m are the numbers of nodes in the i and j directions in block m.
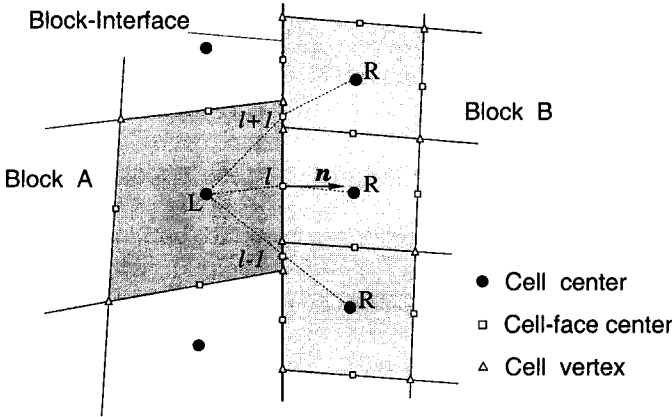
Fig. 8.11. Interface between two blocks with non-matching grids, showing interface CV-faces and the nomenclature

The grids in two neighboring blocks need not match at the interface; an example is shown in Fig. 8.11. Some authors use so called "hanging nodes" on either side of the interface as boundary nodes of each block; we shall describe another possibility. Rather than having hanging nodes we allow CVs along interfaces to have more than four (in 3D more than six) faces.

Since the shaded CV in block A of Fig. 8.11 has three neighbors on its east face, we cannot use the usual notation for structured grids here. This face is not of the regular type (with one neighbor on the opposite side), so we shall not include it while working in block A. The coefficient matrix and the source term for this CV will thus be incomplete, since the contribution from its east side is missing; in particular, the coefficient A_{E} will be zero.

In order to treat the irregular cell faces found at block interfaces, we have to use another kind of data structure here – one similar to the one used when the whole grid is unstructured. Each piece of the interface common to two CVs must be identified (by a pre-processing tool) and placed on a list together with all of the information needed to approximate the surface integrals: the indices of the left (L) and right (R) neighbor cells, the surface vector (pointing from L to R) and the coordinates of cell-face center. With this information, one can use the method used in the interior of each block to approximate the fluxes through these faces. The same approach can be used at the "cuts" that occur in O- and C-type grids; in this case, we are dealing

with an interface between two sides of the same block (i.e., A and B are the same block, but the grids may not match).

Each interface cell face contributes to the source terms for the neighboring CVs (explicit contributions to the convective and diffusive fluxes treated by deferred correction), to the main diagonal coefficient (A_{P}) of these CVs, and to two off-diagonal coefficients: A_L for node R and A_R for node L. The problem of irregularity of data structure due to having three east neighbors is thus overcome by regarding the contributions to the global coefficient matrix as belonging to the interface cell faces (which always have two neighbor cells) rather than to the CVs. It is then irrelevant how the blocks are ordered relative to each other (the east side of one block can be connected to any side of the other block): one has only to provide the indices of the neighbor CVs to the interface cell faces.

The contributions from interface cell faces, namely A_L and A_R, make the global coefficient matrix A irregular: neither the number of elements per row nor the bandwidth is constant. However, this is easily dealt with. All we need to do is to modify the iteration matrix M (see Chap. 5) so that it does not contain the elements due to the faces on the block interfaces. We shall describe a solution algorithm based on an ILU-type of solver; it is easily adapted to other linear equation solvers.

1. Assemble the elements of matrix A and the source term Q in each block, ignoring the contributions of the block interfaces.
2. Loop over the list of interface cell faces, updating A_{P} and Q_{P} at nodes L and R, and calculate the matrix elements stored at the cell face, A_L and A_R.
3. Calculate elements of matrices L and U in each block disregarding neighbor blocks, i.e. as if they were on their own.
4. Calculate the residuals in each block using the regular part of the matrix A (A_{E}, A_{W}, A_{N}, A_{S}, A_{P}, and Q_{P}); the residuals for the CVs along block interfaces are incomplete, since the coefficients that refer to neighbor blocks are zero.
5. Loop over the list of interface cell faces and update the residuals at nodes L and R by adding the products $A_R\phi_R$ and $A_L\phi_L$, respectively; once all faces have been visited, all the residuals are complete.
6. Compute the variable update at each node in each block and return to step 1.
7. Repeat until the convergence criterion is met.

Since the matrix elements referring to nodes in neighbor blocks do not contribute to the iteration matrix M, one expects that the number of iterations required to converge will be larger than it is in the single block case. This effect can be studied by artificially splitting a structured grid into several sub-domains and treating each piece as a block. As already mentioned, this is what is done when implicit methods are parallelized by using domain decomposition in space (see Chap. 11); the degradation of convergence rate

of the linear equation solver is then called *numerical inefficiency.* Schreck and Perić (1993) and Seidl et al. (1996), among others, have performed numerous tests and found that the performance – especially when conjugate gradients and multigrid solvers are used – remains very good even for a large number of sub-domains. When a good structured grid can be constructed, it should be used. Block-structuring does increase the computing effort, but it allows solution of more complex problems and it certainly requires a more complicated algorithm.

An example of the application of this approach is presented in Sect. 8.11. An implementation of the algorithm for O- and C-type grids is found in the code `caffa.f` in directory `2dgl`; see Appendix A.1. Further details of the implementation for block-structured non-matching grids is available in Lilek et al. (1997b).

8.6.6 Unstructured Grids

Unstructured grids allow great flexibility in adapting the grid to domain boundaries. In general, control volumes of arbitrary shape, i.e. with any number of cell faces, can be used. However, grids with mixed CV types are not common; usually, triangles or quadrilaterals are used in 2D and tetrahedra or hexahedra in 3D. Prisms, pyramids, and tetrahedra may be considered special cases of hexahedra so nominally hexahedral grids may include CVs with less than six faces.

The data structure depends on CVs used. The main objects are the CVs and cell vertices. When a grid is generated, a list of vertices is created. Each CV is defined by four or eight vertices, so the list of CVs also contains a list of associated vertices. The order of the vertices in the list represents the relative positions of the cell faces; e.g., first four vertices of a hexahedral CV define the bottom face and the last four the top face, see Fig. 8.12. The positions of the six neighbor CVs is also implicitly defined; e.g., the bottom face defined by vertices 1, 2, 3 and 4 is common to neighbor CV number 1, etc. This is usually adopted in order to reduce the number of arrays necessary for the definition of connectivity between CVs.

One needs also to create a list of cell faces. Such a list is easily defined once the list of CVs and vertices exists, since each face of a CV appears exactly once in another CV, i.e. if all CVs are scanned, the faces defined by the same vertices appear twice. It is only important that the vertices that define a cell face are always ordered in either clockwise or counter-clockwise order. The same information is contained in the list of faces as that described in preceding section for block interfaces.

Another possibility is to introduce object oriented data structure and define objects *vertex, edge, face* and *volume.* Edges are defined by the vertices on either end, faces by lists of edges (which must form a closed polygon), and volumes by lists of faces. The discretization requires approximations to surface and volume integrals; it makes sense to compute these separately.
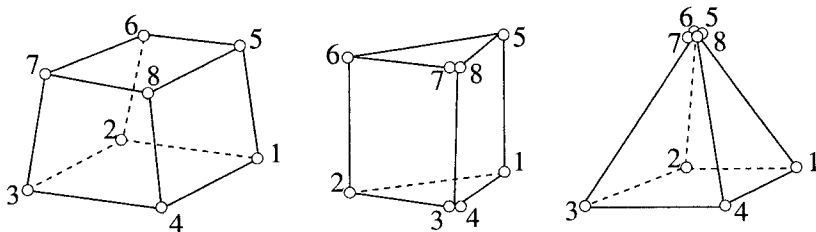
Fig. 8.12. Definition of nominally hexahedral CVs by a list of eight vertices

The data which needs to be stored for each face or volume depends on the integration, differentiation, and interpolation approximations used. We shall not go into details of specific arrangements here, as there are numerous possibilities. Details can be found in books on finite elements, since unstructured grids are the rule rather than the exception in FE methods.

Irregular unstructured grids made up of CVs with more than six faces (polyhedral CVs) are produced when the grid is refined by dividing CVs into smaller ones. In this case, some faces of non-refined CVs are also subdivided into smaller faces. Such a refinement interface can be treated in the same way discussed above for non-matching interfaces between blocks. More details of this kind of local mesh refinement will be given in Chap. 11.

8.7 Control-Volume-Based Finite Element Methods

We give here only a short description of the hybrid FE/FV method using triangular elements and linear shape functions. For more details on finite element methods and their application to Navier-Stokes equations, see books by Oden (1972), Zinkiewicz (1977), Chung (1978), Baker (1983), Girault and Raviart (1986) or Fletcher (1991).
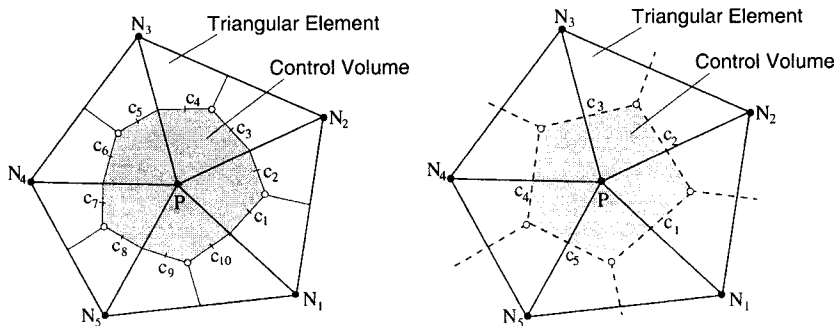
Fig. 8.13. On the principles of control-volume-based finite element and dual-mesh methods

In this method, the solution domain is subdivided into triangular elements. The elements are used to describe the variation of the variables. The computational nodes are located at their vertices. Any variable ϕ is assumed to vary linearly within the element, i.e. its shape function is:

$$\phi = ax + by + c \,. \tag{8.46}$$

The coefficients a, b and c are determined by fitting the function to the nodal values at the vertices. They are thus functions of coordinates and variable values at the nodes.

The control volumes are formed around each node by joining the centroids of the elements and midpoints on element edges, as shown in Fig. 8.13. The conservation equations in integral form are applied to these CVs as described above for the finite volume method. The surface and volume integrals are calculated *element-wise*: for the CV shown in Fig. 8.13, the CV surface consists of 10 sub-faces, and its volume consists of five sub-volumes (from five elements which contribute to the CV). Since the variation of variables over an element is prescribed in form of an analytical function, the integrals can easily be calculated.

The algebraic equation for a CV involves the node P and its immediate neighbors (N_1 to N_5 in Fig. 8.13). Even though the grid consists of triangles only, the number of neighbors varies in general from one CV to another, leading to irregular matrix structure. This restricts the range of solvers which can be used; conjugate gradient and Gauss-Seidel solvers are usually employed.

This approach was followed – although only in 2D and using second order approximations – by Baliga and Patankar (1983), Schneider and Raw (1987), Masson et al. (1994), Baliga (1997), and others. Its extension to 3D is straightforward, but more complicated.

Actually, one does not have to prescribe the shape functions on the elements. Instead, they can be just used to define polyhedral CVs made up of more than one element. For example, tetrahedral grids are easy to generate, but often have bad properties near walls; tetrahedra can be merged to create a polyhedral CV around each tetrahedron vertex as described above. If one adopts the methods of numerical integration, interpolation, and differentiation described earlier, one can lump the sub-faces common to two CVs to form one cell face, thus reducing the number of faces and the computing effort in evaluating surface integrals.

This is sometimes called *dual-mesh approach*. An example is shown in Fig. 8.13, where the resulting CV can be compared to that obtained from the method described above. The number of faces of a cell is now equal to the number of neighbor CVs. If the underlying triangular mesh is highly nonuniform, the lines connecting node P and its neighbors will not pass through the cell-face center. However, this can be handled using approach described in Fig. 8.9 and in Eq. (8.36).

8.8 Pressure-Correction Equation

The SIMPLE-algorithm (see Sect. 7.5.2) needs to be modified when the grid is non-orthogonal and/or unstructured. The approach is described in this section.

For any grid type, the discretized momentum equations have the following form:

$$A_{\mathrm{P}}^{u_i} u_{i,\mathrm{P}} + \sum_l A_l^{u_i} u_{i,l} = Q_{i,\mathrm{P}} \; . \tag{8.47}$$

The source term $Q_{i,\mathrm{P}}$ contains the discretized pressure gradient term. Irrespective of how this term is approximated, one can write:

$$Q_{i,\mathrm{P}} = Q_{i,\mathrm{P}}^{*} + Q_{i,\mathrm{P}}^{p} = Q_{i,\mathrm{P}}^{*} - \left(\frac{\delta p}{\delta x_i}\right)_{\mathrm{P}} \Delta\Omega \; . \tag{8.48}$$

If the pressure term is approximated in a conservative way (as a sum of surface forces), the mean pressure gradient over the CV can be expressed as:

$$Q_{i,\mathrm{P}}^{p} = -\int_S p\, \boldsymbol{i}\cdot\boldsymbol{n}\,\mathrm{d}S = -\int_\Omega \frac{\partial p}{\partial x_i}\,\mathrm{d}\Omega \quad \Rightarrow \quad \left(\frac{\delta p}{\delta x_i}\right)_{\mathrm{P}} = -\frac{Q_{i,\mathrm{P}}^{p}}{\Delta\Omega} \; . \tag{8.49}$$

As always, the correction takes the form of a pressure gradient and the pressure is derived from a Poisson-like equation obtained by imposing the continuity constraint. The objective is to satisfy continuity i.e. the net mass flux into every CV must be zero. In order to calculate the mass flux, we need velocities at the cell face centers. In a staggered arrangement these are available. On colocated grids, they are obtained by interpolation.

It was shown in Chap. 7 that, when interpolated velocities at cell faces are used to derive the pressure-correction equation, a large computational molecule results as can oscillations in the pressure and/or velocities. We described a way to modify the interpolated velocity that yields a compact pressure-correction equation and avoids oscillatory solutions. We shall describe briefly an extension of the approach presented in Sect. 7.5.2 to non-orthogonal grids. The method described below is valid for both conservative and non-conservative treatment of the pressure gradient terms in the momentum equations, and with a little modification can be applied to FD schemes on non-orthogonal grids. It is also valid for arbitrarily shaped CVs, although we shall consider the 'e' face of a regular CV.

The interpolated cell face velocity is corrected by subtracting the difference between the pressure gradient and the interpolated gradient at the cell face location:

$$u_{i,\mathrm{e}}^{m*} = \overline{(u_i^{m*})}_{\mathrm{e}} - \Delta\Omega_{\mathrm{e}} \overline{\left(\frac{1}{A_{\mathrm{P}}^{u_i}}\right)}_{\mathrm{e}} \left[\left(\frac{\delta p}{\delta x_i}\right)_{\mathrm{e}} - \overline{\left(\frac{\delta p}{\delta x_i}\right)}_{\mathrm{e}}\right]^{m-1} , \tag{8.50}$$

where m is the iteration counter. It was shown in Sect. 7.5.2 that, for a 2D uniform grid, the correction corresponds to a central difference approximation to the third derivative to the pressure multiplied by $(\Delta x)^2$; it detects oscillations and smoothes them out. The correction term may be small and not fulfilling its role if A_P is too large. This can happen when unsteady problems are solved using very small time steps, since A_P contains $\Delta\Omega/\Delta t$, which is rarely the case. The correction term may be multiplied by a constant without affecting the consistency of the approximation. This approach to pressure-velocity coupling on colocated grids was developed in early 1980s and is usually attributed to Rhie and Chow (1983). It is widely used and is employed in many commercial CFD codes.

Only the normal velocity component contributes to the mass flux through a cell face. It depends on the pressure gradient in the normal direction. This allows us to write the following expression for the normal velocity component $v_n = \boldsymbol{v} \cdot \boldsymbol{n}$ at a cell face:

$$v_{n,\mathrm{e}}^{m*} = \overline{(v_n^{m*})}_\mathrm{e} - \Delta\Omega_\mathrm{e} \overline{\left(\frac{1}{A_\mathrm{P}^{v_n}}\right)}_\mathrm{e} \left[\left(\frac{\delta p}{\delta n}\right)_\mathrm{e} - \overline{\left(\frac{\delta p}{\delta n}\right)}_\mathrm{e} \right]^{m-1} . \tag{8.51}$$

Since $A_\mathrm{P}^{u_i}$ is the same for all velocity components in a given CV (except near some boundaries), one can replace $A_\mathrm{P}^{v_n}$ by this quantity.

One can calculate the derivative of pressure in the direction normal to 'e' face at the neighboring CV centers and interpolate it to the cell face center. Calculation of the normal derivative at the cell face directly would require a coordinate transformation, which is the usual procedure on structured grids. When using CVs of arbitrary shape, we would like to avoid use of coordinate transformations. Using shape functions is a possibility, but it results in a complex pressure-correction equation. The deferred-correction approach could be used to reduce the complexity.

Another approach can be constructed. From Fig. 8.9, we see that the pressure derivative with respect to n can be approximated as a central difference:

$$\left(\frac{\delta p}{\delta n}\right)_\mathrm{e} \approx \frac{p_{\mathrm{E}'} - p_{\mathrm{P}'}}{|\boldsymbol{r}_{\mathrm{E}'} - \boldsymbol{r}_{\mathrm{P}'}|} = \frac{p_{\mathrm{E}'} - p_{\mathrm{P}'}}{(\boldsymbol{r}_\mathrm{E} - \boldsymbol{r}_\mathrm{P}) \cdot \boldsymbol{n}} . \tag{8.52}$$

The locations of the auxiliary nodes P′ and E′ are easily found:

$$\boldsymbol{r}_{\mathrm{P}'} = \boldsymbol{r}_\mathrm{e} - [(\boldsymbol{r}_\mathrm{e} - \boldsymbol{r}_\mathrm{P}) \cdot \boldsymbol{n}]\,\boldsymbol{n} \, ; \quad \boldsymbol{r}_{\mathrm{E}'} = \boldsymbol{r}_\mathrm{e} - [(\boldsymbol{r}_\mathrm{e} - \boldsymbol{r}_\mathrm{E}) \cdot \boldsymbol{n}]\,\boldsymbol{n} \, . \tag{8.53}$$

These expressions are valid for a CV of any shape. The values of pressure at the two auxiliary nodes can be calculated using cell-center values and gradients:

$$\begin{aligned} p_{\mathrm{P}'} &\approx p_\mathrm{P} + (\mathrm{grad}\, p)_\mathrm{P} \cdot (\boldsymbol{r}_{\mathrm{P}'} - \boldsymbol{r}_\mathrm{P}) \, , \\ p_{\mathrm{E}'} &\approx p_\mathrm{E} + (\mathrm{grad}\, p)_\mathrm{E} \cdot (\boldsymbol{r}_{\mathrm{E}'} - \boldsymbol{r}_\mathrm{E}) \, . \end{aligned} \tag{8.54}$$

With these expressions, Eq. (8.52) becomes:

$$\left(\frac{\delta p}{\delta n}\right)_{\mathrm{e}} \approx \frac{p_{\mathrm{E}} - p_{\mathrm{P}}}{(\boldsymbol{r}_{\mathrm{E}} - \boldsymbol{r}_{\mathrm{P}}) \cdot \boldsymbol{n}} + \frac{(\operatorname{grad} p)_{\mathrm{E}} \cdot (\boldsymbol{r}_{\mathrm{E}'} - \boldsymbol{r}_{\mathrm{E}}) - (\operatorname{grad} p)_{\mathrm{P}} \cdot (\boldsymbol{r}_{\mathrm{P}'} - \boldsymbol{r}_{\mathrm{P}})}{(\boldsymbol{r}_{\mathrm{E}} - \boldsymbol{r}_{\mathrm{P}}) \cdot \boldsymbol{n}} . \tag{8.55}$$

The second term on the right hand side disappears when the line connecting nodes P and E is orthogonal to the cell face i.e. when P and P′ and E and E′ coincide. If the objective is to prevent pressure oscillations on colocated grids, it is sufficient to use just the first term on the right-hand side of Eq. (8.56), i.e. one can approximate Eq. (8.51) as:

$$v_{n,\mathrm{e}}^{m*} = \overline{(v_n^{m*})}_{\mathrm{e}} - \frac{\Delta\Omega_{\mathrm{e}}}{(\boldsymbol{r}_{\mathrm{E}} - \boldsymbol{r}_{\mathrm{P}}) \cdot \boldsymbol{n}} \overline{\left(\frac{1}{A_{\mathrm{P}}^{v_n}}\right)}_{\mathrm{e}} \left[(p_{\mathrm{E}} - p_{\mathrm{P}}) - \overline{(\operatorname{grad} p)}_{\mathrm{e}} \cdot (\boldsymbol{r}_{\mathrm{E}} - \boldsymbol{r}_{\mathrm{P}}) \right] . \tag{8.56}$$

The correction term in square brackets thus represents the difference between the pressure difference $p_{\mathrm{E}} - p_{\mathrm{P}}$ and the approximation to it calculated using interpolated pressure gradient, $\overline{(\operatorname{grad} p)}_{\mathrm{e}} \cdot (\boldsymbol{r}_{\mathrm{E}} - \boldsymbol{r}_{\mathrm{P}})$. For a smooth pressure distribution, this correction term is small and it tends to zero as the grid is refined. The pressure gradient at the CV centers is available as it was calculated for use in the momentum equations.

These mass fluxes calculated using the interpolated velocity,

$$\dot{m}_{\mathrm{e}}^{m*} = \rho_{\mathrm{e}} v_{n,\mathrm{e}}^{m*} S_{\mathrm{e}} , \tag{8.57}$$

do not satisfy the continuity requirement, so their sum results in a mass source:

$$\sum_c \dot{m}_c^* = \Delta \dot{m} \;\; ; \quad c = \mathrm{e}, \mathrm{w}, \mathrm{n}, \mathrm{s}, \ldots \tag{8.58}$$

which must be made to be zero. The velocities have to be corrected so that mass conservation is satisfied in each CV. In an implicit method it is not necessary to satisfy mass conservation exactly at the end of each outer iteration. Following the method described earlier, we correct the mass fluxes by expressing the velocity correction through the gradient of the pressure correction, thus:

$$\dot{m}_{\mathrm{e}}' = \rho_{\mathrm{e}} v_{n,\mathrm{e}}' S_{\mathrm{e}} \approx -(\rho\, \Delta\Omega\, S)_{\mathrm{e}} \overline{\left(\frac{1}{A_{\mathrm{P}}^{v_n}}\right)}_{\mathrm{e}} \left(\frac{\delta p'}{\delta n}\right)_{\mathrm{e}} \approx -(\rho\, \Delta\Omega\, S)_{\mathrm{e}} \overline{\left(\frac{1}{A_{\mathrm{P}}^{v_n}}\right)}_{\mathrm{e}} \left[\frac{p_{\mathrm{E}}' - p_{\mathrm{P}}'}{(\boldsymbol{r}_{\mathrm{E}} - \boldsymbol{r}_{\mathrm{P}}) \cdot \boldsymbol{n}} - \frac{(\operatorname{grad} p')_{\mathrm{E}} \cdot (\boldsymbol{r}_{\mathrm{E}'} - \boldsymbol{r}_{\mathrm{E}}) - (\operatorname{grad} p')_{\mathrm{P}} \cdot (\boldsymbol{r}_{\mathrm{P}'} - \boldsymbol{r}_{\mathrm{P}})}{(\boldsymbol{r}_{\mathrm{E}} - \boldsymbol{r}_{\mathrm{P}}) \cdot \boldsymbol{n}} \right] . \tag{8.59}$$

If the same approximation is applied at the other CV faces, and it is required that the corrected mass fluxes satisfy the continuity equation:

$$\sum_c \dot{m}'_c + \Delta\dot{m} = 0 \ ; \quad c = \mathrm{e, w, n, s}, \ldots \tag{8.60}$$

we obtain the pressure-correction equation.

The last term on the right hand side of Eq. (8.59) leads to an extended computational molecule in the pressure-correction equation. Since this term is small when the non-orthogonality is not severe, it is common practice to neglect it. When the solution converges, the pressure correction becomes zero so the omission of this term does not affect the solution; however, it does affect the convergence rate. For substantially non-orthogonal grids, one has to use a smaller under-relaxation parameter α_p, see Eq. (7.45).

When the above approximation is used, the pressure-correction equation has the usual form; moreover, its coefficient matrix is symmetric so special solvers for symmetric matrices can be used (e.g. the ICCG solver from the conjugate gradients family, see Chap. 5 and directory `solvers` on publisher's server).

The grid non-orthogonality can be taken into account in the pressure-correction equation iteratively, i.e. by using the deferred-correction approach. One solves first the equation for p' in which the non-orthogonality terms in Eq. (8.59) are neglected. In the second step one corrects the error made in the first step by adding another correction:

$$\dot{m}'_\mathrm{e} + \dot{m}''_\mathrm{e} = -(\rho\,\Delta\Omega\,S)_\mathrm{e}\overline{\left(\frac{1}{A_\mathrm{P}^{v_n}}\right)_\mathrm{e}}\left(\frac{\delta p'}{\delta n} + \frac{\delta p''}{\delta n}\right)_\mathrm{e} \ , \tag{8.61}$$

which – by neglecting the non-orthogonality terms in the second correction p'' but taking them into account for the first correction p' – leads to the following expression for the second mass-flux correction:

$$\dot{m}''_\mathrm{e} = -(\rho\,\Delta\Omega\,S)_\mathrm{e}\overline{\left(\frac{1}{A_\mathrm{P}^{v_n}}\right)_\mathrm{e}}\left[\frac{p''_\mathrm{E} - p''_\mathrm{P}}{(\boldsymbol{r}_\mathrm{E} - \boldsymbol{r}_\mathrm{P})\cdot\boldsymbol{n}} - \right.$$
$$\left.\frac{(\mathrm{grad}\,p')_\mathrm{E}\cdot(\boldsymbol{r}_{\mathrm{E}'} - \boldsymbol{r}_\mathrm{E}) - (\mathrm{grad}\,p')_\mathrm{P}\cdot(\boldsymbol{r}_{\mathrm{P}'} - \boldsymbol{r}_\mathrm{P})}{(\boldsymbol{r}_\mathrm{E} - \boldsymbol{r}_\mathrm{P})\cdot\boldsymbol{n}}\right] \ . \tag{8.62}$$

The second term on the right-hand side can now be explicitly calculated, since p' is available.

Since the corrected fluxes $\dot{m}^* + \dot{m}'$ were already forced to satisfy the continuity equation, it follows that $\sum_c \dot{m}''_c = 0$. This leads to an equation for the second pressure correction p'', which has the same matrix A as the equation for p', but a different right hand side. This can be exploited in some solvers. The source term of the second pressure correction contains the divergence of the explicit parts of $\dot{m}''$.

The correction procedure can be continued, by introducing third, fourth etc. corrections. The additional corrections tend to zero; it is rarely necessary to go beyond the two already described as the pressure-correction equation includes more severe approximations than the non-exact treatment of the effects of grid non-orthogonality.

The inclusion of the second pressure correction has a minor effect on the performance of the algorithm if the grid is nearly orthogonal. However, if the angle between n and ξ is less than 45° in much of the domain, convergence may be slow with only one correction. Strong under-relaxation (adding only 5–10% of p' to p^{m-1}) and reduction of the under-relaxation factors for velocity may help but at a cost in efficiency. With two pressure-correction steps, the performance found on orthogonal grids is obtained.
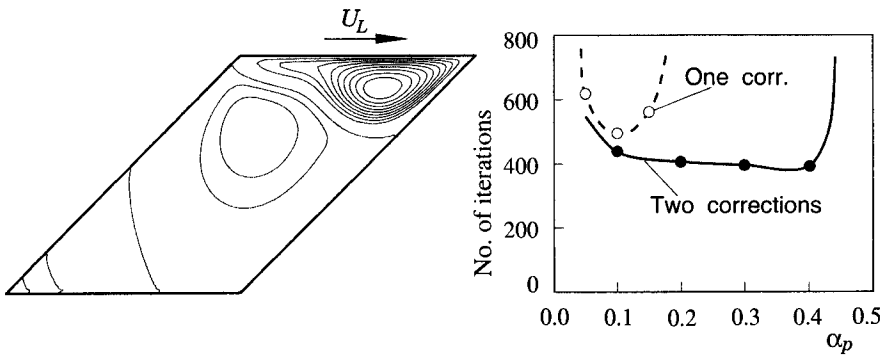
Fig. 8.14. Geometry and predicted streamlines in a lid-driven cavity with side walls inclined at 45°, at Re = 1000 (left), and the numbers of iterations using $\alpha_u = 0.8$ and one or two pressure correction steps, as a function of α_p (right)

An example of performance degradation on non-orthogonal grids without the second correction is shown in Fig. 8.14. Flow in a lid-driven cavity with side walls inclined at 45° was calculated at Re = 1000; Fig. 8.14 also shows the geometry and computed streamlines. The grid lines are parallel to the walls. With the second pressure correction, the numbers of iterations required for convergence and their dependence on the under-relaxation factor for pressure, α_p, are similar to what is found for orthogonal grids, see Fig. 7.14. If the second correction is not included, the range of usable parameter α_p is very narrow and more iterations are required. Similar results are obtained for other values of the under-relaxation factor for velocity α_u, the differences being greater for larger values of α_u. The range of α_p for which convergence is obtained becomes narrower when the angle between grid lines is reduced and only one pressure correction is calculated.

The method described here is implemented in the code found in the directory `2dgl`, see Appendix A.1.

On structured grids, one can transform the normal pressure derivative at cell faces into a combination of derivatives along grid line directions and obtain a pressure-correction equation which involves mixed derivatives; see Sect. 8.5. If the cross-derivatives are treated implicitly, the computational molecule of the pressure-correction equation contains at least nine nodes in 2D and nineteen nodes in 3D. The above two-step procedure results in similar convergence properties as the use of implicitly discretized cross-derivatives (see Perić, 1990), but is computationally more efficient, especially in 3D.

8.9 Axi-Symmetric Problems

Axi-symmetric flows are three-dimensional with respect to Cartesian coordinates i.e. the velocity components are functions of all three coordinates, but they are only two-dimensional in a cylindrical coordinate system (all derivatives with respect to the circumferential direction are zero, and all three velocity components are functions of only the axial and radial coordinates, z and r). In cases without swirl, the circumferential velocity component is zero everywhere. As it is much easier to work with two independent variables than three, for axi-symmetric flows, it makes sense to work in a cylindrical coordinate system rather than a Cartesian one.

In differential form, the 2D conservation equations for mass and momentum, written in a cylindrical coordinate system, read (see e.g. Bird et al., 1962):

$$\frac{\partial \rho}{\partial t} + \frac{\partial(\rho v_z)}{\partial z} + \frac{1}{r}\frac{\partial(\rho r v_r)}{\partial r} = 0 , \tag{8.63}$$

$$\frac{\partial(\rho v_z)}{\partial t} + \frac{\partial(\rho v_z v_z)}{\partial z} + \frac{1}{r}\frac{\partial(\rho r v_r v_z)}{\partial r} = -\frac{\partial p}{\partial z} + \frac{\partial \tau_{zz}}{\partial z} + \frac{1}{r}\frac{\partial(r\tau_{zr})}{\partial r} + \rho b_z , \tag{8.64}$$

$$\frac{\partial(\rho v_r)}{\partial t} + \frac{\partial(\rho v_z v_r)}{\partial z} + \frac{1}{r}\frac{\partial(\rho r v_r v_r)}{\partial r} = -\frac{\partial p}{\partial r} + \frac{\partial \tau_{rz}}{\partial z} + \frac{1}{r}\frac{\partial(r\tau_{rr})}{\partial r} + \frac{\tau_{\theta\theta}}{r} + \frac{\rho v_\theta^2}{r} + \rho b_r , \tag{8.65}$$

$$\frac{\partial(\rho v_\theta)}{\partial t} + \frac{\partial(\rho v_z v_\theta)}{\partial z} + \frac{1}{r}\frac{\partial(\rho r v_r v_\theta)}{\partial r} = -\frac{\rho v_r v_\theta}{r} + \frac{\partial \tau_{\theta z}}{\partial z} + \frac{1}{r^2}\frac{\partial(r^2\tau_{r\theta})}{\partial r} + \rho b_\theta , \tag{8.66}$$

where the non-zero stress tensor components are:

$$\begin{aligned}
\tau_{zz} &= 2\mu\frac{\partial v_z}{\partial z} - \frac{2}{3}\mu\,\mathrm{div}\,\boldsymbol{v}\\
\tau_{rr} &= 2\mu\frac{\partial v_r}{\partial r} - \frac{2}{3}\mu\,\mathrm{div}\,\boldsymbol{v}\\
\tau_{\theta\theta} &= -2\mu\frac{v_r}{r} - \frac{2}{3}\mu\,\mathrm{div}\,\boldsymbol{v}\\
\tau_{rz} &= \tau_{zr} = \mu\left(\frac{\partial v_z}{\partial r} + \frac{\partial v_r}{\partial z}\right)\\
\tau_{\theta r} &= \tau_{r\theta} = \mu r\frac{\partial}{\partial r}\left(\frac{v_\theta}{r}\right)\\
\tau_{\theta z} &= \tau_{z\theta} = \mu\frac{\partial v_\theta}{\partial z}
\end{aligned} \tag{8.67}$$

The above equations contain two terms which have no analog in Cartesian coordinates: $\rho v_\theta^2/r$ in the equation for v_r, which represents the apparent *centrifugal force*, and $\rho v_r v_\theta/r$ in the equation for v_θ, which represents the apparent *Coriolis force*. These terms arise from the coordinate transformation and should not be confused with the centrifugal and Coriolis forces that appear in a rotating coordinate frame. If the swirl velocity v_θ is zero, the apparent forces are zero and the third equation becomes redundant.

When a FD method is used, the derivatives with respect to both axial and radial coordinates are approximated in the same way as in Cartesian coordinates; any method described in Chap. 3 can be used.

Finite volume methods require some care. The conservation equations in integral form given earlier (e.g. (7.79) and (7.80)) remain the same, with the addition of apparent forces as source terms. These are integrated over the volume as described in Sect. 4.3. The CV size in the θ-direction is unity, i.e. one radian. Care is needed with pressure terms. If these are treated as body forces and the pressure derivatives in z and r directions are integrated over the volume as shown in Eqs. (8.40) and (8.41), no additional steps are necessary. However, if the pressure is integrated over the CV surface as in Eq. (8.40), it is not sufficient to integrate only over north, south, west and east cell face, as was the case in plane 2D problems – one has to consider the radial component of pressure forces onto the front and back surfaces.

Thus, we have to add these terms, which have no counterparts in planar 2D problems, to the momentum equation for v_r:

$$Q^r = -\frac{2\mu\,\Delta\Omega}{r_\mathrm{P}^2}v_{r,\mathrm{P}} + p_\mathrm{P}\Delta S + \left(\frac{\rho v_\theta^2}{r}\right)_\mathrm{P}\Delta\Omega\,, \tag{8.68}$$

where ΔS is the area of the front face.

In the equation for v_θ, if it needs to be solved, one has to include the source term (the apparent Coriolis force):

$$Q^\theta = -\left(\frac{\rho v_r v_\theta}{r}\right)_\mathrm{P} \Delta\Omega \,. \tag{8.69}$$

The only other difference compared to planar 2D problems is the calculation of cell face areas and volumes. The areas of cell faces 'n', 'e', 'w' and 's' are calculated as in plane geometry, see Eq. (8.13), with the inclusion of a factor of r_c (where c denotes the cell face center). The areas of the front and back faces are calculated in the same way as the volume in plane geometry (where the third dimension is unity), see Eq. (8.39). The volume of axi-symmetric CVs with any number of faces is:

$$\Delta\Omega = \frac{1}{6}\sum_{i=1}^{N_\mathrm{v}}(z_{i-1} - z_i)(r_{i-1}^2 + r_i^2 + r_i\, r_{i-1}) \,, \tag{8.70}$$

where N_v denotes the number of vertices, counted counter-clockwise, with $i = 0$ corresponding to $i = N_\mathrm{v}$.

An important issue in axi-symmetric swirling flows is the coupling of radial and circumferential velocity components. The equation for v_r contains v_θ^2, and the equation for v_θ the product of v_r and v_θ as source terms, see above. The combination of the sequential (decoupled) solution procedure and Picard linearization may prove inefficient. The coupling can be improved by using the multigrid method for the outer iterations, see Chap. 11, by a coupled solution method, or by using more implicit linearization schemes, see Chap. 5.

If the coordinates z and r of the cylindrical coordinate system are replaced by x and y, the analogy with the equations in Cartesian coordinates becomes obvious. Indeed, if r is set to unity and v_θ and $\tau_{\theta\theta}$ are set to zero, these equations become identical to those in Cartesian coordinates, with $v_z = u_x$ and $v_r = u_y$. Thus the same computer code can be used for both plane and axi-symmetric 2D flows; for axi-symmetric problems, one sets $r = y$ and includes $\tau_{\theta\theta}$ and, if the swirl component is non-zero, the v_θ equation.

An example of application of the FV solution method described above to axi-symmetric problems is shown in Chap. 9.

8.10 Implementation of Boundary Conditions

The implementation of the boundary conditions on non-orthogonal grids requires special attention because the boundaries are usually not aligned with the Cartesian velocity components. The FV method requires that the boundary fluxes either be known or expressed in terms of known quantities and interior nodal values. Of course, the number of CVs must match the number of unknowns.

We shall often refer to a local coordinate system (n, t, s), which is a rotated Cartesian frame with n the outward normal to the boundary and t and s tangential to the boundary.

8.10.1 Inlet

Usually, at an inlet boundary, all quantities have to be prescribed. If the conditions at inlet are not well known, it is useful to move the boundary as far from the region of interest as possible. Since the velocity and other variables are given, all the convective fluxes can be calculated. The diffusive fluxes are usually not known, but they can be approximated using known boundary values of the variables and one-sided finite difference approximations for the gradients.

8.10.2 Outlet

At the outlet we usually know little about the flow. For this reason, these boundaries should be as far downstream of the region of interest as possible. Otherwise, errors may propagate upstream. The flow should be directed out of the domain over the entire outlet cross-section, and if possible, be parallel. In high Reynolds number flows, upstream propagation of errors – at least in steady flows – is weak so it is easy to find approximations for the boundary conditions. Usually one extrapolates along grid lines from the interior to the boundary (or, better, along streamlines). The simplest approximation is that of zero gradient along grid lines. For the convective flux this means that a first order upwind approximation is used. The condition of zero gradient on a grid line can be implemented implicitly. For example, at the east face the first order backward approximation gives $\phi_{\mathrm{E}} = \phi_{\mathrm{P}}$. When we insert this expression into the discretized equation for the CV next to boundary, we have:

$$(A_{\mathrm{P}} + A_{\mathrm{E}})\phi_{\mathrm{P}} + A_{\mathrm{W}}\phi_{\mathrm{W}} + A_{\mathrm{N}}\phi_{\mathrm{N}} + A_{\mathrm{S}}\phi_{\mathrm{S}} = Q_{\mathrm{P}} \ , \tag{8.71}$$

so the boundary value ϕ_{E} does not appear in the equation. This does not mean that the diffusive flux is zero at the outlet boundary, except when the grid is orthogonal to the boundary.

If higher accuracy is required, one has to use higher-order and one-sided finite difference approximations of the derivatives at outlet boundary. Both convective and diffusive fluxes have to be expressed in terms of the variable values at inner nodes.

When the flow is unsteady, especially when turbulence is directly simulated, care is needed to avoid reflection of errors at the outlet boundary. These issues are discussed in Sect. 9.2.

8.10.3 Impermeable Walls

At an impermeable wall, the following condition applies:

$$u_i = u_{i,\mathrm{wall}} \; . \tag{8.72}$$

This condition follows from the fact that viscous fluids sticks to solid boundary (no-slip condition).

Since there is no flow through the wall, convective fluxes of all quantities are zero. Diffusive fluxes require some attention. For scalar quantities, such as thermal energy, they may be zero (adiabatic walls), they may be specified (prescribed heat flux), or the value of the scalar may be prescribed (isothermal walls). If the flux is known, it can be inserted into the conservation equation for the near-wall CVs, e.g. at the south boundary:

$$\int_{S_\mathrm{s}} \Gamma \,\mathrm{grad}\, \phi \cdot \boldsymbol{n} \,\mathrm{d}S = \int_{S_\mathrm{s}} f \,\mathrm{d}S \approx f_\mathrm{s} S_\mathrm{s} \; , \tag{8.73}$$

where f is the prescribed flux per unit area. If the value of ϕ is specified at wall, we need to approximate the normal gradient of ϕ using one-sided differences. From such an approximation we can also calculate the value of ϕ at the wall when the flux is prescribed. Many possibilities exist; one is to calculate the value of ϕ at an auxiliary point P′ located on the normal n, see Fig. 8.15, and use the approximation:

$$\left(\frac{\partial \phi}{\partial n}\right)_\mathrm{s} \approx \frac{\phi_{\mathrm{P}'} - \phi_\mathrm{S}}{\delta n} \; , \tag{8.74}$$

where $\delta n = (\boldsymbol{r}_\mathrm{S} - \boldsymbol{r}_{\mathrm{P}'}) \cdot \boldsymbol{n}$ is the distance between points P′ and S. If the non-orthogonality is not severe, one can use ϕ_P instead of $\phi_{\mathrm{P}'}$. Shape functions or extrapolated gradients from cell centers can also be used.
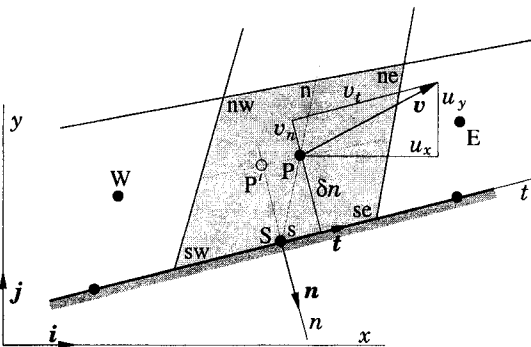
Fig. 8.15. On the implementation of boundary conditions at a wall

Diffusive fluxes in the momentum equations require special attention. If we were solving for the velocity components v_n, v_t and v_s, we could use the approach described in Sect. 7.7. The viscous stresses at a wall are:

$$\tau_{nn} = 2\mu \left(\frac{\partial v_n}{\partial n}\right)_{\text{wall}} = 0 \ , \quad \tau_{nt} = \mu \left(\frac{\partial v_t}{\partial n}\right)_{\text{wall}} \ . \tag{8.75}$$

Here we assume that the coordinate t is in the direction of the shear force at the wall, so $\tau_{ns} = 0$. This force is parallel to the projection of the velocity vector onto the wall (s is orthogonal to it). This is equivalent to the assumption that the velocity vector does not change its direction between the first grid point and the wall, which is not quite true.

Both v_t and v_n can easily be calculated at node P. In 2D, the unit vector $\boldsymbol{t}$ is easily obtained from coordinates of the corners 'se' and 'sw', see Fig. 8.15. In 3D, we have to determine the direction of the vector $\boldsymbol{t}$. From the velocity parallel to the wall we can define $\boldsymbol{t}$ as follows:

$$\boldsymbol{v}_p = \boldsymbol{v} - (\boldsymbol{v} \cdot \boldsymbol{n})\boldsymbol{n} \quad \Rightarrow \quad \boldsymbol{t} = \frac{\boldsymbol{v}_p}{|\boldsymbol{v}_p|} \ . \tag{8.76}$$

The velocity components needed to approximate the stresses are then:

$$v_n = \boldsymbol{v} \cdot \boldsymbol{n} = u n_x + v n_y + w n_z \ , \quad v_t = \boldsymbol{v} \cdot \boldsymbol{t} = u t_x + v t_y + w t_z \ . \tag{8.77}$$

The derivatives can be calculated as in Eq. (8.74).

One could transform the stress τ_{nt} to obtain τ_{xx}, τ_{xy} etc., but this is not necessary. The surface integral of τ_{nt} results in a force:

$$\boldsymbol{f}_{\text{wall}} = \int_{S_s} \boldsymbol{t}\tau_{nt} \, \mathrm{d}S \approx (\boldsymbol{t}\tau_{nt} S)_{\text{s}} \tag{8.78}$$

whose x, y and z components correspond to the integrals needed in the discretized momentum equations; e.g. in 2D we have in the equation for u_x:

$$f_x = \int_{S_s} (\tau_{xx}\boldsymbol{i} + \tau_{yx}\boldsymbol{j} + \tau_{zx}\boldsymbol{k}) \cdot \boldsymbol{n} \, \mathrm{d}S = \boldsymbol{i} \cdot \boldsymbol{f}_{\text{wall}} \approx (t_x \tau_{nt} S)_{\text{s}} \ . \tag{8.79}$$

Alternatively we can use the velocity gradients at cell centers (calculated e.g. using the Gauss theorem, see Eq. (8.20)), extrapolate them to the center of wall cell face, calculate the shear stresses τ_{xx}, τ_{xy} etc., and calculate the shear force components from the above expression.

We thus replace the diffusive fluxes in the momentum equations at walls by the shear force. If this force is calculated explicitly using values from the previous iteration, convergence may be impaired. If the force is written as a function of Cartesian velocity components at node P, part of it can be treated implicitly. In this case the coefficients A_{P} will not be the same for all velocity components, as is the case in the interior cells. This is undesirable, since the coefficients A_{P} are needed in the pressure-correction equation, and if they differ, we would have to store all three values. It is therefore best to use the deferred-correction approach, as in the interior: we approximate

$$f_i^{\text{i}} = \mu S \frac{\delta u_i}{\delta \xi} \ , \tag{8.80}$$

implicitly and add the difference between the implicit approximation and the force calculated using one of the above mentioned approaches to the right hand side of the equation. Here ξ is the local coordinate along the grid line connecting cell face center and node P. The coefficient A_{P} is then the same for all velocity components, and the explicit terms partially cancel out, as discussed in Sect. 7.1. The rate of convergence is almost unaffected.

8.10.4 Symmetry Planes

In many flows there are one or more symmetry planes. When the flow is steady, there is a solution which is symmetric with respect to this plane (in many cases, e.g. diffusers or channels with sudden expansions, there exist also asymmetric steady solutions). The symmetric solution can be obtained by solving the problem in part of the solution domain only using symmetry conditions.

At a symmetry plane the convective fluxes of all quantities are zero. Also, the normal gradients of the velocity components parallel to symmetry plane and of all scalar quantities are zero there. The normal velocity component is zero, but its normal gradient is not; thus, the normal stress τ_{nn} is non-zero. The surface integral of τ_{nn} results in a force:

$$\boldsymbol{f}_{\mathrm{sym}} = \int_{S_{\mathrm{s}}} \boldsymbol{n}\tau_{nn}\,\mathrm{d}S \approx (\boldsymbol{n}\tau_{nn}S)_{\mathrm{s}} \,. \tag{8.81}$$

When the symmetry boundary does not coincide with a Cartesian coordinate plane, the diffusive fluxes of all three Cartesian velocity components will be non-zero. These fluxes can be calculated by obtaining first the resultant normal force from (8.81) and an approximation of the normal derivative as described in the preceding section, and splitting this force into Cartesian components. Alternatively, one can extrapolate the velocity gradients from interior to the boundary and use an expression similar to (8.79), e.g. for the u_x component at the face 's' (see Fig. 8.15):

$$f_x = \int_{S_{\mathrm{s}}} (\tau_{xx}\boldsymbol{i} + \tau_{yx}\boldsymbol{j} + \tau_{zx}\boldsymbol{k}) \cdot \boldsymbol{n}\,\mathrm{d}S = \boldsymbol{i} \cdot \boldsymbol{f}_{\mathrm{sym}} \approx (n_x\tau_{nn}S)_{\mathrm{s}} \,. \tag{8.82}$$

As in the case of wall boundaries, one can split the diffusive fluxes at a symmetry boundary into an implicit part, involving velocity component at the CV center (which contributes to the coefficient A_{P}) or use the deferred-correction approach to keep A_{P} same for all velocity components.

8.10.5 Specified Pressure

In incompressible flows one usually specifies the mass flow rate at inlet and uses extrapolation at the outlet. However, there are situations in which the

mass flow rate is not known, but the pressures at the inlet and outlet are prescribed. Also, pressure is sometimes specified at a far field boundary.

When the pressure is specified at a boundary, velocity cannot be prescribed – it has to be extrapolated from the interior using the same approach as for cell faces between two CVs, see Eq. (8.51). The pressure gradient at the boundary is approximated using one-sided differences; for example, at the 'e' face, one can use expression (8.26), which is a first order backward difference.

The boundary velocities determined in this way need to be corrected to satisfy the mass conservation; the mass flux corrections $\dot{m}'$ are not zero at boundaries where the pressure is specified. However, boundary pressure is not corrected, i.e. $p' = 0$ at boundary. This is used as a Dirichlet boundary condition in the pressure-correction equation.

If the Reynolds number is high, the solution process will converge slowly if the above approach is applied when the inlet and outlet pressures are specified. Another possibility is to guess first the mass flow rate at inlet and treat it as prescribed for one outer iteration, and consider the pressure to be specified only at outlet. The inlet velocities should then be corrected by trying to match the extrapolated pressure at the inlet boundary with the specified pressure. An iterative correction procedure is used to drive the difference between two pressures to zero.

8.11 Examples

As an example we consider laminar flow around a circular cylinder in a channel, see Fig. 8.16. At the inlet, a parabolic velocity profile is prescribed:

$$u_x = \frac{6U}{H^2}\left[(y - y_\mathrm{B})H - (y - y_\mathrm{B})^2\right] \; , \quad u_y = 0 \; , \tag{8.83}$$

where U is the mean velocity, $H = 4.1\,D$ is the channel height and $y_\mathrm{B} = -2\,D$ is the y-coordinate of the bottom wall. The axis of the cylinder is not on the horizontal symmetry plane of the channel so the flow is slightly asymmetric.

Steady flow at a Reynolds number Re = 20, based on the mean velocity in the channel and cylinder diameter, is considered first. Calculations were performed on four systematically refined block-structured grids with non-matching interfaces; the level two grid is shown in Fig. 8.17. The first grid had 1250 CVs, and the finest had 80 000 CVs. Second order CDS was used for spatial discretization. The forces on cylinder were the quantities of primary interest.

If the configuration were fully symmetric, the steady flow would yield zero lift force on cylinder. Due to the asymmetry, there is a small lift force, as the flow rate (and therefore the pressure) is different above and below the cylinder. The drag and lift coefficients are defined as:
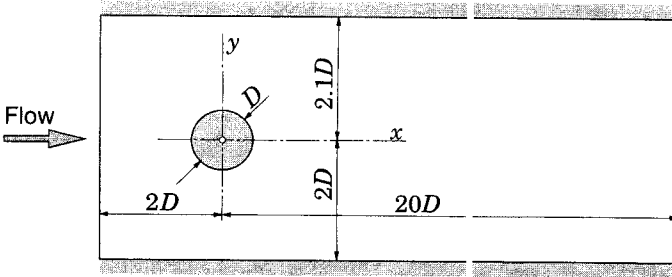
Fig. 8.16. Geometry and boundary conditions for laminar flow around a circular cylinder in a channel
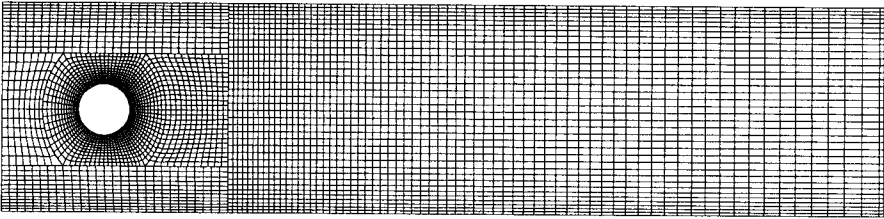
Fig. 8.17. The level two grid, used to calculate 2D flow around a circular cylinder in a channel (5000 CVs; only part of the grid is shown)
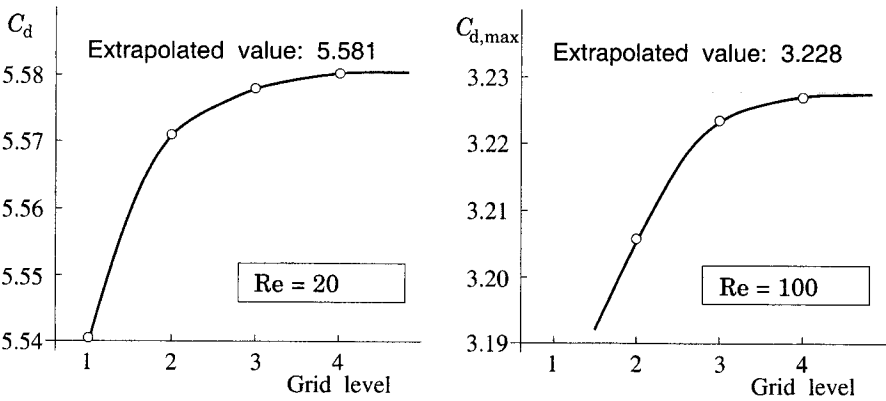
Fig. 8.18. Drag coefficients for the 2D flow around a cylinder in a channel as functions of grid size: steady flow at Re = 20 (left) and the maximum drag coefficient in a periodic unsteady flow at Re = 100 (right); from Muzaferija et al. (1995)

$$C_\mathrm{d} = \frac{f_x}{\frac{1}{2}\rho U^2} , \quad C_\mathrm{l} = \frac{f_y}{\frac{1}{2}\rho U^2} , \tag{8.84}$$

where f_x and f_y are the x and y component of the force exerted by the fluid on the cylinder. This force is calculated by integrating the pressure and

shear force over the cylinder surface. Values obtained using results on all four grids are shown in Fig. 8.18; the value obtained using Richardson extrapolation is also indicated. Second order convergence is obtained, as expected. The discretization error on the finest grid was approximately 0.02%. The lift coefficient converges in the same way; its extrapolated value is $C_l = 0.0105$. The lift coefficient is thus about 530 times smaller than the drag coefficient.
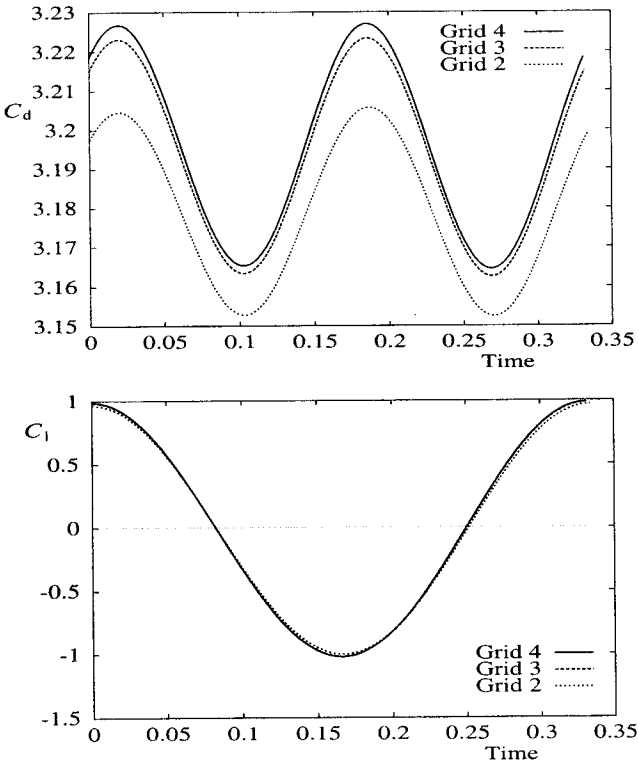
Fig. 8.19. Variation of the coefficients of lift (above) and drag (below) on a cylinder in a channel at Re = 100, as functions of time (results on the three finest grids are shown); from Muzaferija et al. (1995)

When the Reynolds number is increased beyond a critical value (which for a cylinder in infinite stream is about 40), the flow becomes unsteady and vortices are shed from the cylinder. Flow at Re = 100 was investigated by Muzaferija et al. (1995). A second-order three time-level implicit scheme was used for time integration. Starting impulsively from rest, the flow goes through a development stage and eventually becomes periodic. Due to vortex shedding, both the drag and lift forces oscillate. In a symmetric configuration the lift coefficient would oscillate around zero; in this case, however, it

oscillates between $C_{\rm l,min} = -1.021$ and $C_{\rm l,max} = 0.987$. The drag coefficient oscillates between $C_{\rm d,min} = 3.165$ and $C_{\rm d,max} = 3.228$. The convergence of the drag coefficient as the grid is refined is shown in Fig. 8.18. The time step was the same for all grids; there are 663 time steps per oscillation period of the lift force. Calculations with larger time steps (twice and four times larger) showed very little dependence of the result on the time step size for any given grid; the spatial discretization errors are much larger than the temporal discretization errors.
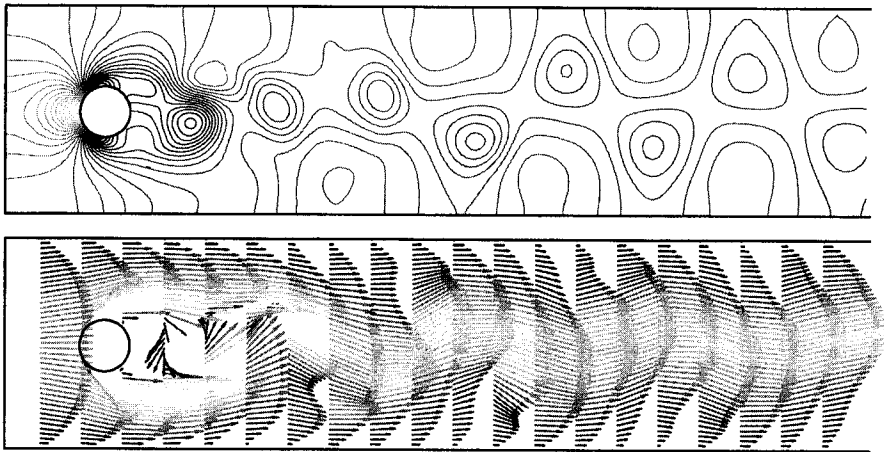
Fig. 8.20. Instantaneous isobars (above) and velocity vectors (below) in the laminar 2D flow around a circular cylinder in a channel at Re = 100; from Muzaferija et al. (1995)

The drag and lift forces oscillate at different frequencies: the drag has twice the frequency as the lift. The reason is that the drag force has one maximum and one minimum during the growth and shedding of each vortex, while the sign of the lift force depends on the location of the vortex i.e. whether it is above or below the cylinder. The Strouhal number, defined as

$$\mathrm{St} = \frac{D}{UT} , \tag{8.85}$$

where T is the oscillation period in $C_{\rm l}$ (corresponding to the inverse frequency of the vortex shedding), was found to be 0.3018. This value is much higher than for a cylinder in an infinite stream (0.18 – 0.2); the confinement in the channel speeds up the the processes associated with vortex shedding.

The oscillations are also shifted in phase by about 10% of the drag oscillation period. The variation of $C_{\rm d}$ and $C_{\rm l}$ over one lift period is shown in Fig. 8.19, where results for the three finest grids are presented; the scale for $C_{\rm d}$

has been enlarged to emphasize the differences between the solutions. The expected second order convergence towards grid-independence can be seen. A further increase in Reynolds number would make the pattern more and more irregular, eventually the flow becomes turbulent.

Figure 8.20 shows instantaneous isobars and velocity vectors. The closed pressure contours indicate the locations of vortex centers, where the pressure has a local minimum.
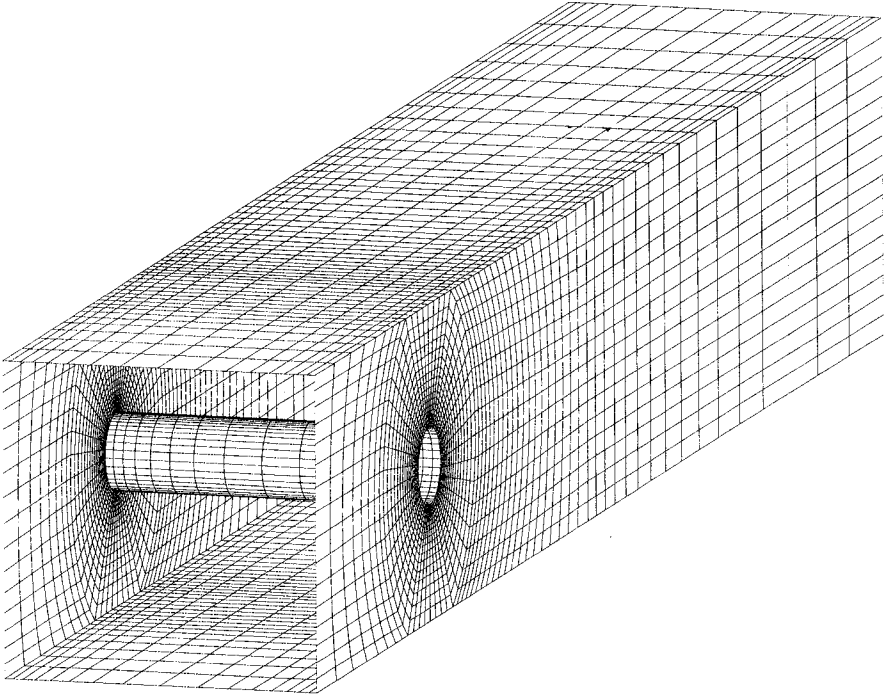
Fig. 8.21. The level two grid used to calculate 3D flow around a circular cylinder in a channel with square cross-section (23 552 CV; only part of grid is shown)

Muzaferija et al. (1995) performed also calculations of 3D laminar flow around a circular cylinder mounted between two walls of a square channel. The cross-sectional configuration is the same as in the 2D case, see Fig. 8.21, but the inlet section was $5D$ long. The profile for fully-developed laminar flow in a square channel was prescribed at inlet. For the same mean velocity, the velocity at the channel centerline is much higher than in 2D case ($2.25\,U$ in place of $1.5\,U$). The boundary layers at the side walls affect the drag and lift forces per unit cylinder length; they are higher in the 3D configuration.

Finally, Fig. 8.22 shows pressure distribution on the cylinder surface and two surfaces of constant pressure, calculated on a grid with 188 416 CV at Re = 100. This figure highlights the difficulties of visualizing 3D flows. Velocity
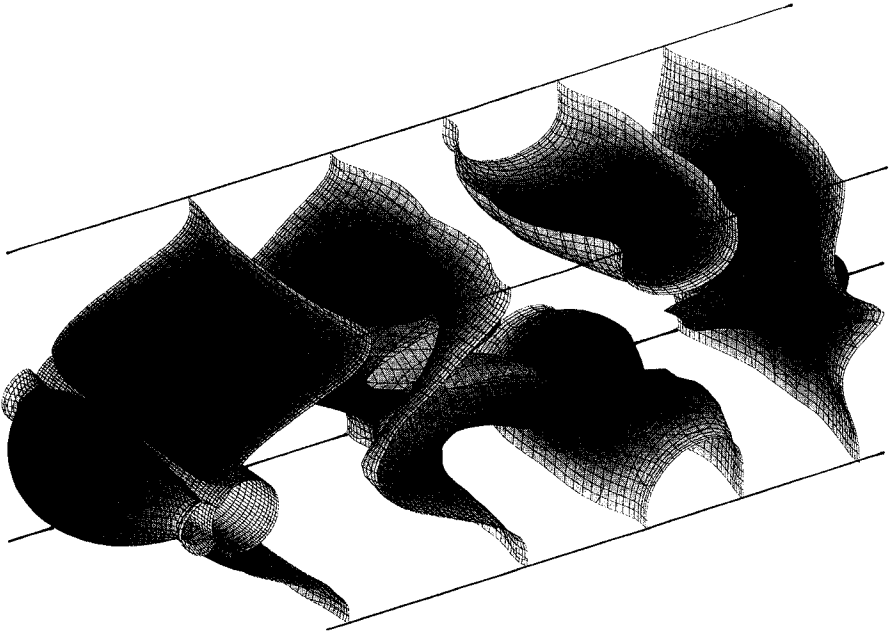
Fig. 8.22. Pressure distribution on the cylinder surface and the surfaces of constant pressure $p = 1.1$ (in front of the cylinder) and $p = 0.3$ (behind the cylinder) for the flow over a cylinder in a square channel, calculated on a grid with 188 416 CVs at Re = 100 (pressure surface 'colored' by the magnitude of the u_x velocity component); from Muzaferija et al. (1995)

vectors and streamlines, which are often used in 2D, are difficult to both draw and interpret in 3D problems. Presentation of contours and vector projections on selected surfaces (planes, iso-surfaces of some quantity, boundary surfaces etc.) and the possibility to view them from different directions is perhaps the best way of analyzing 3D flows. Unsteady flows require animation of the results. We shall not deal further with these issues, but want to stress their importance.

9. Turbulent Flows

9.1 Introduction

Most flows encountered in engineering practice are turbulent and therefore require different treatment. Turbulent flows are characterized by the following properties:

- Turbulent flows are highly unsteady. A plot of the velocity as a function of time at most points in the flow would appear random to an observer unfamiliar with these flows. The word 'chaotic' could be used but it has been given another definition in recent years.
- They are three-dimensional. The time-averaged velocity may be a function of only two coordinates, but the instantaneous field fluctuates rapidly in all three spatial dimensions.
- They contain a great deal of vorticity. Indeed, vortex stretching is one of the principal mechanisms by which the intensity of turbulence is increased.
- Turbulence increases the rate at which conserved quantities are stirred. Stirring is a process in which parcels of fluid with differing concentrations of at least one of the conserved properties are brought into contact. The actual *mixing* is accomplished by diffusion. Nonetheless, this process is often called *turbulent diffusion.*
- By means of the processes just mentioned, turbulence brings fluids of differing momentum content into contact. The reduction of the velocity gradients due to the action of viscosity reduces the kinetic energy of the flow; in other words, mixing is a *dissipative* process. The lost energy is irreversibly converted into internal energy of the fluid.
- It has been shown in recent years that turbulent flows contain *coherent structures*—repeatable and essentially deterministic events that are responsible for a large part of the mixing. However, the random component of turbulent flows causes these events to differ from each other in size, strength, and time interval between occurrences, making study of them very difficult.
- Turbulent flows fluctuate on a broad range of length and time scales. This property makes direct numerical simulation of turbulent flows very difficult. (See below.)

All of these properties are important. The effects produced by turbulence may or may not be desirable, depending on the application. Intense mixing

is useful when chemical mixing or heat transfer are needed; both of these may be increased by orders of magnitude by turbulence. On the other hand, increased mixing of momentum results in increased frictional forces, thus increasing the power required to pump a fluid or to propel a vehicle; again, an increase by an order of magnitude is not unusual. Engineers need to be able to understand and predict these effects in order to achieve good designs. In some cases, it is possible to control the turbulence, at least in part.

In the past, the primary approach to studying turbulent flows was experimental. Overall parameters such as the time-averaged drag or heat transfer are relatively easy to measure but as the sophistication of engineering devices increases, the levels of detail and accuracy required also increase, as does cost and the expense and difficulty of making measurements. To optimize a design, it is usually necessary to understand the source of the undesired effects; this requires detailed measurements that are costly and time-consuming. Some types of measurements, for example, the fluctuating pressure within a flow, are almost impossible to make at the present time. Others cannot be made with the required precision. As a result, numerical methods have an important role to play.

Before proceeding to the discussion of numerical methods for these flows, it is useful to introduce a classification scheme for the approaches to predicting turbulent flows. According to Bardina et al. (1980) there are six categories, most of which can be divided in sub-categories.

- The first involves the use of *correlations* such as ones that give the friction factor as a function of the Reynolds number or the Nusselt number of heat transfer as a function of the Reynolds and Prandtl numbers. This method, which is usually taught in introductory courses, is very useful but is limited to simple types of flows, ones that can be characterized by just a few parameters. As its use does not require the use of a computer, we shall say no more about it here.
- The second uses *integral equations* which can be derived from the equations of motion by integrating over one or more coordinates. Usually this reduces the problem to one or more ordinary differential equations which are easily solved. The methods applied to these equations are those for ordinary differential equations which are discussed in Chap. 6.
- The third is based on equations obtained by averaging the equations of motion over time (if the flow is statistically steady), over a coordinate in which the mean flow does not vary, or over an ensemble of realizations (an imagined set of flows in which all controllable factors are kept fixed). This approach is called *one-point closure* and leads to a set of partial differential equations called the *Reynolds-averaged Navier-Stokes* (or RANS) equations. As we shall see later, these equations do not form a closed set so this method requires the introduction of approximations (*turbulence models*). Some of the turbulence models in common use today and a discussion

of the problems associated with the numerical solution of equations containing turbulence models are presented later in this chapter.
- The fourth type of method is called *two-point closure*. It uses equations for the correlation of the velocity components at two spatial points or, more often, the Fourier transform of these equations. As these methods are rarely used except for homogeneous turbulence, we shall not consider them further.
- The fifth is *large eddy simulation (LES)* and solves for the largest scale motions of the flow while approximating or modeling only the small scale motions. It can be regarded as a kind of compromise between one point closure methods (see above) and direct numerical simulation (see below).
- Finally, there is *direct numerical simulation (DNS)* in which the Navier-Stokes equations are solved for all of the motions in a turbulent flow.

As one progresses down this list, more and more of the turbulent motions are computed and fewer are approximated by models. This makes the methods close to the bottom more exact but the computation time is increased considerably.

All of the methods described in this chapter require the solution of some form of the conservation equations for mass, momentum, energy, or chemical species. The major difficulty is that turbulent flows contain variations on a much wider range of length and time scales than laminar flows. So, even though they are similar to the laminar flow equations, the equations describing turbulent flows are usually much more difficult and expensive to solve.

9.2 Direct Numerical Simulation (DNS)

The most accurate approach to turbulence simulation is to solve the Navier-Stokes equations without averaging or approximation other than numerical discretizations whose errors can be estimated and controlled. It is also the simplest approach from the conceptual point of view. In such simulations, all of the motions contained in the flow are resolved. The computed flow field obtained is equivalent to a single realization of a flow or a short-duration laboratory experiment; as noted above, this approach is called direct numerical simulation (DNS).

In a direct numerical simulation, in order to assure that all of the significant structures of the turbulence have been captured, the domain on which the computation is performed must be at least as large as the physical domain to be considered or the largest turbulent eddy. A useful measure of the latter scale is the integral scale (L) of the turbulence which is essentially the distance over which the fluctuating component of the velocity remains correlated. Thus, each linear dimension of the domain must be at least a few

times the integral scale. A valid simulation must also capture all of the kinetic energy dissipation. This occurs on the smallest scales, the ones on which viscosity is active, so the size of the grid must be no larger than a viscously determined scale, called the Kolmogoroff scale, η.

For homogeneous isotropic turbulence, the simplest type of turbulence, there is no reason to use anything other than a uniform grid. In this case, the argument just given shows that number of grid points in each direction must be at least L/η; it can be shown (Tennekes and Lumley, 1976) that this ratio is proportional to $\mathrm{Re}_L^{3/4}$. Here Re_L is a Reynolds number based on the magnitude of the velocity fluctuations and the integral scale; this parameter is typically about 0.01 times the macroscopic Reynolds number engineers use to describe a flow. Since this number of points must be employed in each of the three coordinate directions, and the time step is related to the grid size, the cost of a simulation scales as Re_L^3. In terms of the Reynolds number that an engineer would use to describe the flow, the scaling of the cost may be somewhat different.

Since the number of grid points that can be used in a computation is limited by the processing speed and memory of the machine on which it is carried out, direct numerical simulation is possible only for flows at relatively low Reynolds numbers and in geometrically simple domains. On present machines, it is possible to make direct numerical simulations of homogeneous flows at turbulent Reynolds numbers up to a few hundred. As noted in the preceding paragraph, this corresponds to overall flow Reynolds numbers about two orders of magnitude larger and allows DNS to reach the low end of the range of Reynolds numbers of engineering interest, making it a useful method in some cases. In other cases, it may be possible to extrapolate from the Reynolds number of the simulation to the Reynolds number of actual interest by using some kind of extrapolation. For further details about DNS, see the recent review by Leonard (1995).

The results of a DNS contain very detailed information about the flow. This can be very useful but, on the one hand, it is far more information than any engineer needs and, on the other, DNS is too expensive to be employed very often and cannot be used as a design tool. One must then ask what DNS can be used for. With it, we can obtain detailed information about the velocity, pressure, and any other variable of interest at a large number of grid points. These results may be regarded as the equivalent of experimental data and can be used to produce statistical information or to create a 'numerical flow visualization.' From the latter, one can learn a great deal about the coherent structures that exist in the flow. This wealth of information can then be used to develop a qualitative understanding of the physics of the flow or to construct a quantitative model, perhaps of the RANS type, which will allow other, similar, flows to be computed.

We thus conclude that the major role that DNS can fill is as a research tool. Some examples of kinds of uses to which DNS has been put are:

- Understanding the mechanisms of turbulence production, energy transfer, and dissipation in turbulent flows;
- Simulation of the production of aerodynamic noise;
- Understanding the effects of compressibility on turbulence;
- Understanding the interaction between combustion and turbulence;
- Controlling and reducing drag on a solid surface.

Other applications of DNS have already been made and many others will undoubtedly be proposed in the future.

The increasing speed of computers has made it possible to carry out DNS of simple flows at low Reynolds numbers on workstations. By simple flows, we mean any homogeneous turbulent flow (there are many), channel flow, free shear flows, and a few others. On large parallel computers, it is now possible to do DNS with 512^3 ($\sim 1.35 \times 10^8$) or more grid points. The computation time depends on the machine and the number of grid points used so no useful estimate can be given. Indeed, one usually chooses the flow to simulate and the number of grid points to fit the available computer resources. A complete state of the art simulation generally requires between ten and many hundred hours. As computers become faster and memories larger, more complex and higher Reynolds number flows will be simulated.

A wide variety of numerical methods can be employed in direct numerical simulation and large eddy simulation. Almost any method described in this book can be used. Because these methods have been presented in earlier chapters, we shall not give a lot of detail here. However, there are important differences between DNS and LES and simulations of steady flows and it is important that these be discussed.

The most important requirements placed on numerical methods for DNS and LES arise from the need to produce an accurate realization of a flow that contains a wide range of length and time scales. Because an accurate time history is required, techniques designed for steady flows are inefficient and should not be used without considerable modification. The need for accuracy requires the time step to be small and, obviously, the time-advance method must be stable for the time step selected. In most cases, explicit methods that are stable for the time step demanded by the accuracy requirement are available so there is no reason to incur the extra expense associated with implicit methods; most simulations have therefore used explicit time advance methods. A notable (but not the only) exception occurs near solid surfaces. The important structures in these regions are of very small size and very fine grids must be used, especially in the direction normal to the wall. Numerical instability may arise from the viscous terms involving derivatives normal to the wall so these are often treated implicitly. In complex geometries, it may be necessary to treat still more terms implicitly.

The time advance methods most commonly used in DNS and LES are of second to fourth order accuracy; Runge-Kutta methods have been used most commonly but others, such as the Adams-Bashforth and leapfrog methods

have also been used. In general, for a given order of accuracy, Runge-Kutta methods require more computation per time step. Despite this, they are preferred because, for a given time step, the errors they produce are much smaller than those of the competing methods. Thus, in practice, they allow a larger time step for the same accuracy and this more than compensates for the increased amount of computation. The Crank-Nicolson method is often applied to the terms that must be treated implicitly.

A difficulty with time-advance methods is that ones of accuracy higher than first order require storage of data at more than one time step (including intermediate time steps) and, as the amount of data contained in a single velocity field is large, the demand for storage may exceed what is available even with the large memories in modern computers. This puts a premium on designing and using methods which demand relatively little storage. As an example of a method of this kind, Leonard and Wray (1982) presented a third order Runge-Kutta method which requires less storage than the standard Runge-Kutta method of that accuracy.

A further issue of importance in DNS is the need to handle a wide range of length scales; this requires a change in the way one thinks about discretization methods. The most common descriptor of the accuracy of a spatial discretization method is its order, a number that describes the rate at which the discretization error decreases when the grid size is reduced. Some discussion of why this is not the appropriate measure of quality was given in Chap. 3; we shall amplify on it here. Again, it is useful to think in terms of the Fourier decomposition of the velocity field. We showed (Sect. 3.10) that, on a uniform grid, the velocity field can be represented in terms of a Fourier series:

$$u(x) = \sum \tilde{u}(k)\,\mathrm{e}^{\mathrm{i}kx} \tag{9.1}$$

The highest wavenumber k that can be resolved on a grid of size Δx is $\pi/\Delta x$, so we consider only $0 < k < \pi/\Delta x$. The series (9.1) can be differentiated term by term. The exact derivative of $\mathrm{e}^{\mathrm{i}kx}$, $\mathrm{i}k\mathrm{e}^{\mathrm{i}kx}$ is replaced by $\mathrm{i}k_{\mathrm{eff}}\mathrm{e}^{\mathrm{i}kx}$ where k_{eff} is the effective wavenumber defined in Sect. 3.10 when a finite difference approximation is used. The plot of k_{eff} given in Fig. 3.6 shows that central differences are accurate only for $k < \pi/2\Delta x$, the first half of the wavenumber range of interest.

The difficulty for turbulent flow simulations that is not encountered in steady flow simulations is that turbulence spectra (the distributions of turbulence energy over wavenumber or inverse length scale) are usually large over a significant part of the wavenumber range $\{0, \pi/\Delta x\}$ so the order of the method, which measures the accuracy of the approximation at low wavenumber, is no longer a good measure of accuracy. A better measure of error is:

$$\epsilon_1^2 = \frac{\displaystyle\int (k - k_{\text{eff}})^2 E(k)\,\mathrm{d}k}{\displaystyle\int k^2 E(k)\,\mathrm{d}k}\,, \tag{9.2}$$

where $E(k)$ is the energy spectrum of the turbulence which, in one dimension, is $\hat{u}(k)\hat{u}^*(k)/2$, where the asterisk indicates complex conjugation. Similar expressions can be given for the second derivative. Using the measure (9.2), Cain et al. (1981) found that, for a spectrum typical of isotropic turbulence, a fourth order central difference method had half the error of a second order method, a much larger fraction than one would anticipate.

It is also useful to reiterate the importance of using an energy-conservative spatial differencing scheme. Many methods, including all upwind ones, are dissipative; that is, they include as part of the truncation error a diffusive term that dissipates energy in a time-dependent calculation. Their use has been advocated because the dissipation they introduce often stabilizes numerical methods. When these methods are applied to steady problems, the dissipative error may not be too large in the steady-state result (although we showed in earlier chapters that these errors may be quite large). When these methods are used in DNS, the dissipation produced is often much greater than that due to the physical viscosity and the results obtained may have little connection to the physics of the problem. For a discussion of energy conservation, see section 7.1.3. Also, as demonstrated in Chap. 7, energy conservation prevents the velocity from growing without bound and thus maintains stability.

The methods and step sizes in time and space need to be related. The errors made in the spatial and temporal discretizations should be as nearly equal as possible i.e. they should be balanced. This is not possible point-by-point and for every time step but, if this condition is not satisfied in an average sense, one is using too fine a step in one of the independent variables and the simulation could be made at lower cost with little loss of accuracy.

Accuracy is difficult to measure in DNS and LES. The reason is inherent in the nature of turbulent flows. A small change in the initial state of a turbulent flow is amplified exponentially in time and, after a relatively short time, the perturbed flow hardly resembles the original one. This is a physical phenomenon that has nothing to do with the numerical method. Since any numerical method introduces some error and any change in the method or the parameters will change that error, direct comparison of two solutions with the goal of determining the error is not possible. Instead, one can repeat the simulation with a different grid (which should differ considerably from the original one) and statistical properties of the two solutions can be compared. From the difference, an estimate of the error can be found. Unfortunately, it is difficult to know how the error changes with the grid size, so this type of estimate can only be an approximation. A simpler approach, which has been used by most people who compute simple turbulent flows, is to look at the spectrum of the turbulence. If the energy in the smallest scales is sufficiently

smaller than that at the peak in the energy spectrum, it is probably safe to assume that the flow has been well resolved.

The accuracy requirement makes use of spectral methods common in DNS and LES. These methods were described briefly earlier, in Sect. 3.10. In essence, they use Fourier series as a means of computing derivatives. The use of Fourier transforms is feasible only because the fast Fourier transform algorithm (Cooley and Tukey, 1965) reduces the cost of computing a Fourier transform to $n \log_2 n$ operations. Unfortunately, this algorithm is applicable only for equi-spaced grids and a few other special cases. A number of specialized methods of this kind have been developed for solving the Navier-Stokes equations; the reader interested in more details of spectral methods is referred to the book by Canuto et al. (1987).

We briefly mention one special type of method. Rather than directly approximating the Navier-Stokes equations, one could multiply them by a sequence of 'test functions' and integrate over the entire domain and then find a solution that satisfies the resulting equations. This procedure is similar to one used in deriving finite element methods. Functions which satisfy this form of the equations are known as 'weak solutions'. One can represent the solution of the Navier-Stokes equations as a series of vector functions, each of which has zero divergence. This choice removes the pressure from the integral form of the equations, thereby reducing the number of dependent variables that need to be computed and stored. The set of dependent variables can be further reduced by noting that, if a function has zero divergence, its third component can be computed from the other two. The result is that only two sets of dependent variables need to be computed, reducing the memory requirements by half. As these methods are quite specialized and their development requires considerable space, they are not given in detail here; the interested reader is referred to the paper by Moser et al. (1983).

Another difficulty in DNS is that of generating initial and boundary conditions. The former must contain all the details of the initial three-dimensional velocity field. Since coherent structures are an important component of the flow, it is difficult to construct such a field. Furthermore, the effects of initial conditions are typically 'remembered' by the flow for a considerable time, usually a few 'eddy-turnover times.' An eddy-turnover time is essentially the integral time scale of the flow or the integral length scale divided by the root-mean-square velocity (q). Thus the initial conditions have a significant effect on the results. Frequently, the first part of a simulation that is started with artificially constructed initial conditions must be discarded because it is not faithful to the physics. The question of how to select initial conditions is as much art as science and no unique prescriptions applicable to all flows can be given but we shall give some examples.

For homogeneous isotropic turbulence, the simplest case, periodic boundary conditions are used and it is easiest to construct the initial conditions in Fourier space i.e., we need to create $\hat{u}_i(\boldsymbol{k})$. This is done by giving the

spectrum which sets the amplitude of the Fourier mode i.e., $|\hat{u}_i(\boldsymbol{k})|$. The requirement of continuity $\boldsymbol{k} \cdot \hat{u}_i(\boldsymbol{k})$ places another restriction on that mode. This leaves just one random number to be chosen to completely define $\hat{u}_i(\boldsymbol{k})$; it is usually a phase angle. The simulation must then be run for about two eddy turnover times before it can be considered to represent real turbulence.

The best initial conditions for other flows are obtained from the results of previous simulations. For example, for homogeneous turbulence subjected to strain, the best initial conditions are taken from developed isotropic turbulence. For channel flow, the best choice has been found to be a mixture of the mean velocity, instability modes (which have nearly the right structure), and noise. For a curved channel, one can take the results of a fully developed plane channel flow as the initial condition.

Similar considerations apply to the boundary conditions where the flow enters the domain (inflow conditions). The correct conditions must contain the complete velocity field on a plane (or other surface) of a turbulent flow at each time step which is difficult to construct. As an example, one way this can be done for the developing flow in a curved channel is to use results for the flow in a plane channel. A simulation of a plane channel flow is made (either simultaneously or in advance) and the velocity components on one plane normal to the main flow direction provide the inflow condition for the curved channel.

As already noted, for flows which do not vary (in the statistical sense) in a given direction, one can use periodic boundary conditions in that direction. These are easy to use, fit especially well with spectral methods, and provide conditions at the nominal boundary that are as realistic as possible.

Outflow boundaries are less difficult to handle. One possibility is to use extrapolation conditions which require the derivatives of all quantities in the direction normal to the boundary be zero:

$$\frac{\partial \phi}{\partial n} = 0 \, , \tag{9.3}$$

where ϕ is any of the dependent variables. This condition is often used in steady flows but is not satisfactory in unsteady flows. For the latter, it is better to replace this condition by an unsteady convective condition. A number of such conditions have been tried but one that appears to work well is also one of the simplest:

$$\frac{\partial \phi}{\partial t} + U \frac{\partial \phi}{\partial n} = 0 \, , \tag{9.4}$$

where U is a velocity that is independent of location on the outflow surface and is chosen so that overall conservation is maintained i.e., it is the velocity required to make the outflow mass flux equal to the incoming mass flux. This condition appears to avoid the problem caused by pressure perturbations being reflected off the outflow boundary back to the interior of the domain.

On solid walls, no slip boundary conditions, which have been described in Chaps. 7 and 8 may be used. One must bear in mind that at boundaries of this type the turbulence tends to develop small but very important structures ('streaks') that require very fine grids especially in the direction normal the wall and, to a lesser extent, in the spanwise direction (the direction normal to both the wall and the principal flow direction).

Symmetry boundary conditions, which are often used in RANS computations to reduce the size of the domain are usually not applicable in DNS or LES because, although the mean flow may be symmetric about some particular plane, the instantaneous flow is not and important physical effects may be removed by application of conditions of this type. Symmetry conditions have, however, been used to represent free surfaces.

Despite all attempts to make the initial and boundary conditions as realistic as possible, a simulation must be run for some time before the flow develops all of the correct characteristics of the physical flow. This situation derives from the physics of turbulent flows so there is little one can do to speed up the process; one possibility is mentioned below. As we have noted, the eddy turnover time scale is the key time scale of the problem. In many flows, it can be related to a time scale characteristic of the flow as a whole i.e. a mean flow time scale. However, in separated flows, there are regions that communicate with the remainder of the flow on a very long time scale and the development process can be very slow, making very long run times necessary.

The best way to ascertain that flow development is complete is to monitor some quantity, preferably one that is sensitive to the parts of the flow that are slow to develop; the choice depends on the flow being simulated. As an example, one might measure a spatial average of the skin friction in the recirculating region of a separated flow as a function of time. Initially, there is usually a systematic increase or decrease of the monitored quantity; when the flow becomes fully developed, the value will show statistical fluctuations with time. After this point, statistical average results (for example, for the mean velocity or its fluctuations) may be obtained by averaging over time and/or a statistically homogeneous coordinate in the flow. In so doing it is important to remember that, because turbulence is not purely random, the sample size is not the same as the number of points used in the averaging process. A conservative estimate is to assume that each volume of diameter equal to the integral scale (and each time period equal to the integral time scale) represents only a single sample.

The development process can be sped up by using a coarse grid initially. When the flow is developed on that grid, the fine grid can be introduced. If this is done, some waiting is still necessary for the flow to develop on the fine grid but it may be smaller than the time that would have been required had the fine grid been used throughout the simulation.

9.2.1 Example: Spatial Decay of Grid Turbulence

As an illustrative example of what can be accomplished with DNS, we shall take a deceptively simple flow, the flow created by an oscillating grid in a large body of quiescent fluid. The oscillation of the grid creates turbulence which decreases in intensity with distance from the grid. This process of energy transfer away from the oscillating grid is usually called ***turbulent diffusion***; energy transfer by turbulence plays an important role in many flows so its prediction is important but it is surprisingly difficult to model. Briggs et al. (1996) made simulations of this flow and obtained good agreement with the experimentally determined rate of decay of the turbulence with distance from the grid. The energy decays approximately as $x^{-\alpha}$ with $2 < \alpha < 3$; determination of the exponent α is difficult both experimentally and computationally because the rapid decay does not provide a large enough region to allow one to compute its value accurately.
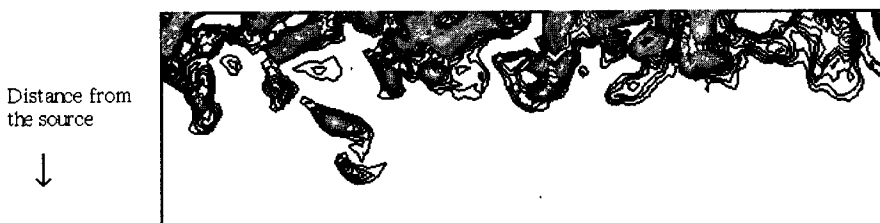
Fig. 9.1. Contours of the kinetic energy on a plane in the flow created by an oscillating grid in a quiescent fluid; the grid is located at the top of the figure. Energetic packets of fluid transfer energy away from the grid region. From Briggs et al. (1996)

Using visualizations based on simulations of this flow, Briggs et al. (1996) showed that the dominant mechanism of turbulent diffusion in this flow is the movement of energetic parcels of fluid through the undisturbed fluid. This may seem a simple and logical explanation but is contrary to earlier proposals. Figure 9.1 shows the contours of the kinetic energy on one plane in this flow. One sees that the large energetic regions are of approximately the same size throughout the flow but there are fewer of them far from the grid. The reasons are that those parcels that propagate parallel to the grid do not move very far in the direction normal to the grid and that small 'blobs' of energetic fluid are quickly destroyed by the action of viscous diffusion.

The results were used to test turbulence models. A typical example of such a test is shown in Fig. 9.2 in which the profile to the flux of turbulent kinetic energy is given and compared with the predictions of some commonly used turbulence models. It is clear that the models do not work very well even in a flow as simple as this one. The probable reason is that the models were

designed to deal with turbulence generated by shear which has a character very different from that created by the oscillating grid.
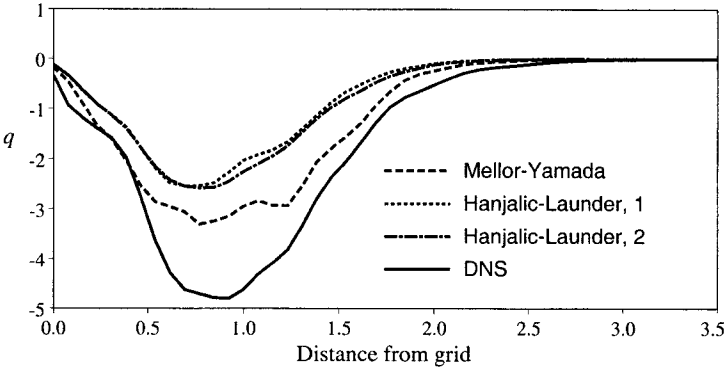
Fig. 9.2. The profile of the flux of turbulent kinetic energy, q, compared with the predictions of some commonly used turbulence models (Mellor and Yamada, 1982; Hanjalić and Launder, 1976 and 1980); from Briggs et al. (1996)

The simulation used a code that was designed for the simulation of homogeneous turbulence (Rogallo, 1981). Periodic boundary conditions are applied in all three directions; this implies that there is actually a periodic array of grids but this causes no problem so long as the distance between neighboring grids is sufficiently greater than the distance required for the turbulence to decay. The code uses the Fourier spectral method with periodic boundary conditions in all three spatial directions and a third-order Runge-Kutta method in time.

These results illustrate some important features of DNS. The method allows one to compute statistical quantities that can be compared with experimental data to validate the results. It also allows computation of quantities that are difficult to measure in the laboratory and that are useful in assessing models. At the same time, the method yields visualizations of the flow that can provide insight into the physics of the turbulence. It is rarely possible to obtain both statistical data and visualizations of the same flow in a laboratory. As the example above shows, the combination can be very valuable.

In direct numerical simulations, one can control the external variables in a manner that is difficult or impossible to implement in the laboratory. There have been several cases in which the results produced by DNS disagreed with those of experiments and the former turned out to be more nearly correct. One example is the distribution of turbulent statistics near a wall in a channel flow; the results of Kim et al. (1987) proved to be more accurate than the experiments when both were repeated with more care. An earlier example was provided by Bardina et al. (1980) which explained some apparently

anomalous results in an experiment on the effects of rotation on isotropic turbulence.

DNS makes it possible to investigate certain effects much more accurately than would be otherwise possible. It is also possible to try methods of control that cannot be realized experimentally. The point of doing so is to provide insight into the physics of the flow and thus to indicate possibilities that may be realizable (and to point the direction toward realizable approaches). An example is the study of drag reduction and control on a flat plate conducted by Choi et al. (1994). They showed that, by using controlled blowing and suction through the wall (or a pulsating wall surface), the turbulent drag of a flat plate could be reduced by 30%. Bewley (1994) used optimal control methods to demonstrate the possibility that the flow could be forced to re-laminarize at low Reynolds number and that reduction in the skin friction is possible at high Reynolds numbers.

9.3 Large Eddy Simulation (LES)

As we have noted, turbulent flows contain a wide range of length and time scales; the range of eddy sizes that might be found in a flow is shown schematically on the left-hand side of Fig. 9.3. The right-hand side of this figure shows the time history of a typical velocity component at a point in the flow; the range of scales on which fluctuations occur is obvious.
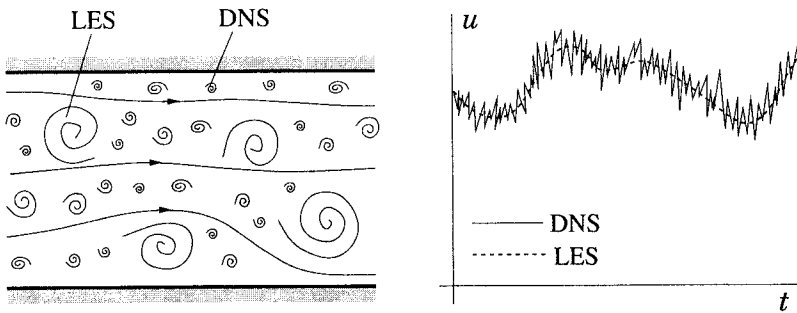
Fig. 9.3. Schematic representation of turbulent motion (left) and the time dependence of a velocity component at a point (right)

The large scale motions are generally much more energetic than the small scale ones; their size and strength make them by far the most effective transporters of the conserved properties. The small scales are usually much weaker and provide little transport of these properties. A simulation which treats the large eddies more exactly than the small ones may make sense; large eddy simulation is just such an approach. Large eddy simulations are three dimensional, time dependent and expensive but much less costly than DNS

of the same flow. In general, because it is more accurate, DNS is the preferred method whenever it is feasible. LES is the preferred method for flows in which the Reynolds number is too high or the geometry is too complex to allow application of DNS.

It is essential to define the quantities to be computed precisely. We need a velocity field that contains only the large scale components of the total field. This is best produced by filtering the velocity field (Leonard, 1974); in this approach, the large or resolved scale field, the one to be simulated, is essentially a local average of the complete field. We shall use one-dimensional notation; the generalization to three dimensions is straightforward. The filtered velocity is defined by:

$$\overline{u}_i(x) = \int G(x,x')\, u_i(x')\, \mathrm{d}x' \;, \tag{9.5}$$

where $G(x,x')$, the filter kernel, is a localized function. Filter kernels which have been applied in LES include a Gaussian, a box filter (a simple local average) and a cutoff (a filter which eliminates all Fourier coefficients belonging to wavenumbers above a cutoff). Every filter has a length scale associated with it, Δ. Roughly, eddies of size larger than Δ are large eddies while those smaller than Δ are small eddies, the ones that need to be modeled.

When the Navier-Stokes equations with constant density (incompressible flow) are filtered, one obtains a set of equations very similar to the RANS equations:

$$\frac{\partial(\rho\overline{u}_i)}{\partial t} + \frac{\partial(\rho\overline{u_i u_j})}{\partial x_j} = -\frac{\partial \overline{p}}{\partial x_i} + \frac{\partial}{\partial x_j}\left[\mu\left(\frac{\partial \overline{u}_i}{\partial x_j} + \frac{\partial \overline{u}_j}{\partial x_i}\right)\right] \;. \tag{9.6}$$

Since the continuity equation is linear, filtering does not change it:

$$\frac{\partial(\rho\overline{u}_i)}{\partial x_i} = 0 \;. \tag{9.7}$$

It is important to note that, since

$$\overline{u_i u_j} \neq \overline{u}_i \overline{u}_j \tag{9.8}$$

and the quantity on the left side of this inequality is not easily computed, a modeling approximation for the difference between the two sides of this inequality,

$$\tau^{\mathrm{s}}_{ij} = -\rho(\overline{u_i u_j} - \overline{u}_i \overline{u}_j) \tag{9.9}$$

must be introduced. In the context of LES, τ^{s}_{ij} is called the *subgrid-scale Reynolds stress.* The name 'stress' stems from the way in which it is treated rather than its physical nature. It is in fact the large scale momentum flux caused by the action of the small or unresolved scales. The name 'subgrid scale' is also somewhat of a misnomer. The width of the filter, Δ, need not

have anything to do with the grid size, h, other than the obvious condition that $\Delta > h$. Some authors do make such a connection and their nomenclature has stuck. The models used to approximate the SGS Reynolds stress (9.9) are called *subgrid-scale (SGS)* or *subfilter-scale models.*

The subgrid-scale Reynolds stress contains local averages of the small scale field so models for it should be based on the local velocity field or, perhaps, on the past history of the local fluid. The latter can be accomplished by using a model that solves partial differential equations to obtain the parameters needed to determine the SGS Reynolds stress.

9.3.1 Smagorinsky and Related Models

The earliest and most commonly used subgrid scale model is one proposed by Smagorinsky (1963). It is an eddy viscosity model. All such models are based on the notion that the principal effects of the SGS Reynolds stress are increased transport and dissipation. As these phenomena are due to the viscosity in laminar flows, it seems reasonable to assume that a reasonable model might be:

$$\tau^{\mathrm{s}}_{ij} - \frac{1}{3}\tau^{\mathrm{s}}_{kk}\delta_{ij} = \mu_{\mathrm{t}}\left(\frac{\partial \overline{u}_i}{\partial x_j} + \frac{\partial \overline{u}_j}{\partial x_i}\right) = 2\mu_{\mathrm{t}}\overline{S}_{ij} \ , \tag{9.10}$$

where μ_{t} is the eddy viscosity and $\overline{S}_{ij}$ is the strain rate of the large scale or resolved field. This model can be derived in a number of ways including heuristic methods, for example, by equating production and dissipation of subgrid-scale turbulent kinetic energy, or via turbulence theories. Similar models are also often used in connection with the RANS equations; see below.

The form of the subgrid-scale eddy viscosity can be derived by dimensional arguments and is:

$$\mu_{\mathrm{t}} = C_{\mathrm{S}}^2 \rho \Delta^2 |\overline{S}| \ , \tag{9.11}$$

where C_{S} is a model parameter to be determined, Δ is the filter length scale, and $|\overline{S}| = (\overline{S}_{ij}\overline{S}_{ij})^{1/2}$. This form for the eddy viscosity can be derived in a number of ways. Theories provide estimates of the parameter. Most of these methods apply only to isotropic turbulence for which they all agree that $C_{\mathrm{S}} \approx 0.2$. Unfortunately, C_{S} is not constant; it may be a function of Reynolds number and/or other non-dimensional parameters and may take different values in different flows.

The Smagorinsky model, although relatively successful, is not without problems. To simulate channel flow with it, several modifications are required. The value of the parameter C_{S} in the bulk of the flow has to be reduced from 0.2 to approximately 0.065, which reduces the eddy viscosity by almost an order of magnitude. Changes of this magnitude are required in all shear flows. In regions close to the surfaces of the channel, the value has to be reduced

even further. One successful recipe is to borrow the van Driest damping that has long been used to reduce the near-wall eddy viscosity in RANS models:

$$C_S = C_{S0}\left(1 - e^{-n^+/A^+}\right)^2 , \tag{9.12}$$

where n^+ is the distance from the wall in viscous wall units ($n^+ = nu_\tau/\nu$, where u_τ is the shear velocity, $u_\tau = \sqrt{\tau_w/\rho}$, and τ_w is the shear stress at the wall) and A^+ is a constant usually taken to be approximately 25. Although this modification produces the desired results, it is difficult to justify in the context of LES. An SGS model should depend solely on the local properties of the flow and it is difficult to see how the distance from the wall qualifies in this regard.

The purpose of the van Driest damping is to reduce the subgrid-scale eddy viscosity near the wall; $\mu_t \sim n^3$ in this region and models should respect this property. An alternative is a subgrid-scale model which reduces the eddy viscosity when the subgrid-scale Reynolds number, $|\overline{S}|\Delta^2/\nu$, becomes small. Models of this kind were suggested by McMillan and Ferziger (1980) and by Yakhot and Orszag (1986); the latter used renormalization group theory to derive their model.

A further problem is that, near a wall, the flow structure is very anisotropic. Regions of low and high speed fluid (streaks) are created; they are approximately 1000 viscous units long and 30–50 viscous units wide in both the spanwise and normal directions. Resolving the streaks requires a highly anisotropic grid and the choice of length scale, Δ, to use in the SGS model is not obvious. The usual choice is $(\Delta_1\Delta_2\Delta_3)^{1/3}$ but $(\Delta_1^2 + \Delta_2^2 + \Delta_3^2)^{1/2}$ is possible and others are easily constructed; here Δ_i is the width associated with the filter in the ith coordinate direction. It is possible that, with a proper choice of length scale, the damping (9.12) would become unnecessary. A fuller discussion of this issue can be found in Piomelli et al. (1989).

In a stably-stratified fluid, it is necessary to reduce the Smagorinsky parameter. Stratification is common in geophysical flows; the usual practice is to make the parameter a function of a Richardson or Froude number. These are related non-dimensional parameters that represent the relative importance of stratification and shear. Similar effects occur in flows in which rotation and/or curvature play significant roles. In the past, the Richardson number was based on the properties of the mean flow field. Recent work indicates that it is better to base the parameter on properties of the turbulence than on the applied forces; Ivey and Imberger (1991) suggested using the turbulent Froude number.

Thus there are many difficulties with the Smagorinsky model. If we wish to simulate more complex and/or higher Reynolds number flows, it may be important to have a more accurate model. Indeed, detailed tests based on results derived from DNS data, show that the Smagorinsky model is quite poor in representing the details of the subgrid-scale stresses.

The smallest scales that are resolved in a simulation are similar in many ways to the still smaller scales that are treated via the model. This idea leads to an alternative subgrid-scale model, the *scale-similarity model* (Bardina et al., 1980). The principal argument is that the important interactions between the resolved and unresolved scales involve the smallest eddies of the former and the largest eddies of the latter i.e., eddies that are a little larger or a little smaller than the length scale, Δ, associated with the filter. Arguments based on this concept lead to the following model:

$$\tau^{\mathrm{s}}_{ij} = -\rho(\overline{\overline{u}_i\overline{u}_j} - \overline{\overline{u}}_i\overline{\overline{u}}_j) , \tag{9.13}$$

where the double overline indicates a quantity that has been filtered twice. We have given a more recent version of this model that is Galilean invariant; the original one was not. A constant could be included on the right hand side but it has been found to be very close to unity. This model correlates very well with the actual SGS Reynolds stress, but dissipates hardly any energy and cannot serve as a 'stand alone' SGS model. It transfers energy from the smallest resolved scales to larger scales, which is useful. To correct for the lack of dissipation, it is necessary to combine the Smagorinsky and scale similarity models to produce a 'mixed' model. This model improves the quality of simulations. For further details, see Bardina et al. (1980).

9.3.2 Dynamic Models

The concept underlying the scale similarity model, namely that the smallest resolved scale motions can provide information that can be used to model the largest subgrid scale motions, can be taken a step further, leading to the dynamic model or procedure (Germano et al., 1990). This procedure is based on the assumption that one of the models described above is an acceptable representation of the small scales.

One way to understand the concept behind this method is the following. Suppose we do a large eddy simulation on a fine grid. Let us, for the sake of argument, regard the results as an exact representation of the velocity field. We can then use the following procedure to estimate the subgrid-scale model parameter. The velocity field $\overline{u}_i$ can be filtered (using a filter broader than the one used in the LES itself) to obtain a very large scale field $\overline{\overline{u}}_i$; an effective subgrid-scale field (which actually contains the smallest scales of the simulation being done) can be obtained by subtraction of the two fields. By multiplication and filtering, one can compute the subgrid-scale Reynolds stress tensor produced by that field. From the large-scale field, one can also construct the estimate of this Reynolds stress that the model would produce. By comparing these two, we can test the quality of the model in a direct way and, even more importantly, compute the value of the model parameter. This can be done at every spatial point and every time step. The value of the parameter obtained can then be applied to the subgrid scale model of the

large eddy simulation itself. In this way, a kind of self-consistent subgrid-scale model is produced.

Thus the essential ingredient in this model is the assumption that the same model with the same value of the parameter can be applied to both the actual LES and LES done on a coarser scale. A secondary assumption that is made more for convenience than necessity is that the parameter is independent of location. Finally, we note that the dynamic procedure gives the model parameter as the ratio of two quantities.

The actual procedure of Germano et al. is a bit more formal than what we have just described but the result is the same; the model parameter is computed, at every spatial grid point and every time step, directly from results of the LES itself. We shall not present the formal procedure here. The interested reader is referred to the original paper of Germano et al. (1990) or the review by Ferziger (1995).

This process should be called a procedure rather than a model as any subgrid-scale model can be used as a basis for it. A number of variations are possible. One particularly significant improvement to the original model proposed by Germano et al. (1990) is the least squares procedure suggested by Lilly (1991). The dynamic procedure with the Smagorinsky model as its basis removes many of the difficulties described earlier:

- In shear flows, the Smagorinsky model parameter needs to be much smaller than in isotropic turbulence. The dynamic model produces this change automatically.
- The model parameter has to be reduced even further near walls. The dynamic model automatically decreases the parameter in the correct manner near the wall.
- The definition of the length scale for anisotropic grids or filters is unclear. This issue becomes moot with the dynamic model because the model compensates for any error in the length scale by changing the value of the parameter.

Although it is a considerable improvement on the Smagorinsky model, there are problems with the dynamic procedure. The model parameter it produces is a rapidly varying function of the spatial coordinates and time so the eddy viscosity takes large values of both signs. Although a negative eddy viscosity has been suggested as a way of representing energy transfer from the small scales to the large ones (this process is called *backscatter*), if the eddy viscosity is negative over too large a spatial region or for too long a time, numerical instability can and does occur. One cure is to set any eddy viscosity $\mu_t < -\mu$, the molecular viscosity, equal to $-\mu$; this is called *clipping*. Another useful alternative is to employ averaging in space or time. For details, the reader is referred to the papers cited above. These techniques produce further improvements but are still not completely satisfactory; finding a more robust model for the subgrid scale is the subject of current research.

The arguments on which the dynamic model is based are not restricted to the Smagorinsky model. One could, instead, use the mixed Smagorinsky–scale-similarity model. The mixed model has been used by Zang et al. (1993) and Shah and Ferziger (1995) with considerable success.

Finally, we note that other versions of the dynamic procedure have been devised to overcome the difficulties with the simplest form of the model. One of the better of these is the Lagrangian dynamic model of Meneveau et al (1996). In this model, the terms in the numerator and denominator of the expression for the model parameter terms of the dynamic procedure are averaged along flow trajectories. This is done by solving partial differential equations for these quantities.

The boundary conditions and numerical methods used for LES are very similar to those used in DNS. The most important difference is that, when LES is applied to flows in complex geometries, some numerical methods (for example, spectral methods) become difficult to apply. In these cases, one is forced to use finite difference, finite volume, or finite element methods. In principle, any method described earlier in this book could be used, but it is important to bear in mind that structures that challenge the resolution of the grid may exist almost anywhere in the flow. For this reason, it is important to employ methods of the highest accuracy possible.

In LES, it is possible to use wall functions of the kind used in RANS modeling (see next section). This approach has been shown to work well for attached flows (see Piomelli et al., 1989) but, despite considerable effort, it is not yet known whether this approach can be made to work for separated flows.

It is also important to note that, because LES and DNS require large amounts of computer time, the programs used to make these kinds of simulations are usually special-purpose codes i.e., they are written for a specific geometry and contain special programming elements designed to obtain the highest performance on a particular machine. For this reason, the discretization methods employed in DNS and LES are often particular to the problem being solved.

We should also note that, because the unsteadiness that is inherent to turbulence often affects a flow in profound ways, it is not uncommon to find significant differences in the predictions of the two methods. This introduces the possibility of using a crude form of LES as a tool for determining the gross features of a flow. Indeed, a few industrial applications of this kind have been made.

9.3.3 Deconvolution Models

The most recent approach to SGS modeling is based on the deconvolution concept. These models attempt to estimate the unfiltered velocity from the filtered one. They then use this estimated velocity to compute the subgrid Reynolds stress from its definition (9.9). These models share with the dynamic

procedure the advantage of not requiring any externally provided information such as model constants.

We shall describe a simple version of the model in order to give a flavor of the approach. The unfiltered velocity on the right side of Eq. (9.5) is expanded in a Taylor series about the point x. Truncating the series (usually keeping only terms up to second order) gives a differential equation for the unfiltered velocity in terms of the filtered velocity. We give the result for the simplest case in which the filter kernel is symmetric about the point x. We have:

$$\overline{u_i}(x) = u_i(x) + \frac{\Delta^2}{24}\nabla^2 u_i \ , \tag{9.14}$$

which is the desired differential equation. Shah (1998) used an approximate inversion of this equation that consisted of using approximate factorization (see Chap. 5) and performing just one iteration. He computed several flows with the resulting model, obtaining very good results. Katapodes et al. (2000) used a simpler approximate inversion which consists of simply iterating Eq. (9.14) to get:

$$u_i(x) \approx \overline{u}_i(x) - \frac{\Delta^2}{24}\nabla^2 \overline{u}_i \quad . \tag{9.15}$$

They also preesented more complex versions of the model.

A still more complex, but more accurate, approach to the deconvolution modeling concept has been presented by Domaradzki and coworkers (Domaradzki and Saiki, 1997).

This ends the presentation of subgrid-scale models. At present, reasonable subgrid-scale models exist and they generally produce good simulations. However, the models are not sufficiently precise to be trusted to simulate a flow that has never been treated before. There is need for further improvements and there is a considerable amount of ongoing research.

9.3.4 Example: Flow Over a Wall-Mounted Cube

As an example of the method, we shall use the flow over a cube mounted on one wall of a channel. The geometry is shown in Fig. 9.4. For the simulation shown, which was made by Shah and Ferziger (1997), the Reynolds number based on the maximum velocity at the inflow and the cube height is 3200. The inflow is fully developed channel flow and was taken from a separate simulation of that flow, the outlet condition was the convective condition given above. Periodic boundary conditions were used in the spanwise direction and no-slip conditions at all wall surfaces.

The LES used a grid of $240 \times 128 \times 128$ control volumes with second order accuracy. The time advancement method was of the fractional step type. The convective terms were treated explicitly by a third order Runge-Kutta method in time while the viscous terms were treated implicitly. In particular,
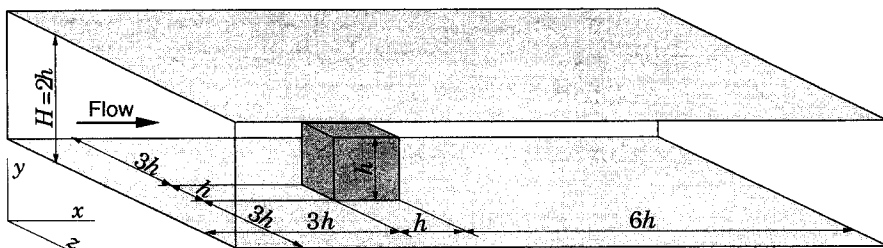
Fig. 9.4. The solution domain for the flow over a cube mounted on a channel wall; from Shah and Ferziger, (1997)

the method used for the latter was an approximate factorization of the Crank-Nicolson method. The pressure was obtained by solving a Poisson equation with the multigrid method.
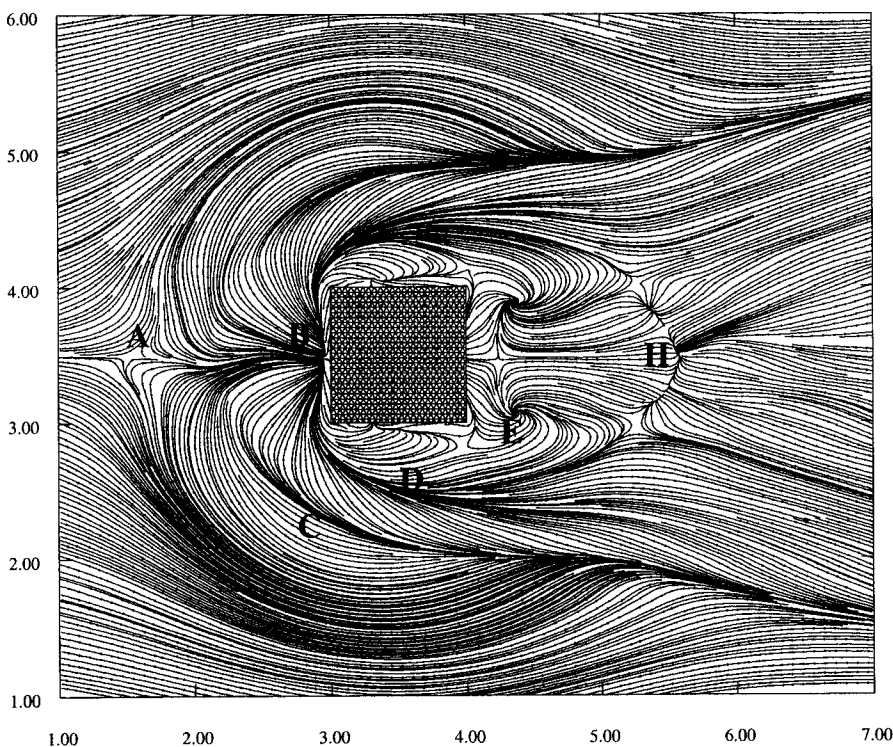
Fig. 9.5. The streamlines in the region close to the lower wall of the flow over a wall-mounted cube; from Shah and Ferziger (1997)

Figure 9.5 gives the streamlines of the time-averaged flow in the region close to the wall; a great deal of information about the flow can be discerned from this plot. The incoming flow does not separate in the traditional sense but reaches a stagnation or saddle point (marked by A on the figure) and goes around the body. Some of the flow further above the lower wall hits the front face of the cube; about half of it flows downwards and creates the region of reversed flow in front of the body. As the flow down the front face of the cube nears the lower wall, there is a secondary separation and a reattachment line (marked by B in the figure) just ahead of the cube. On each side of the cube, one finds a region of converging streamlines (marked as C) and another of diverging streamlines (marked D); these are the traces of the horseshoe vortex (about which more is said below). Behind the body one finds two areas of swirling flow (marked E) which are the footprints of an arch vortex. Finally, there is a reattachment line (marked H) further downstream of the body.
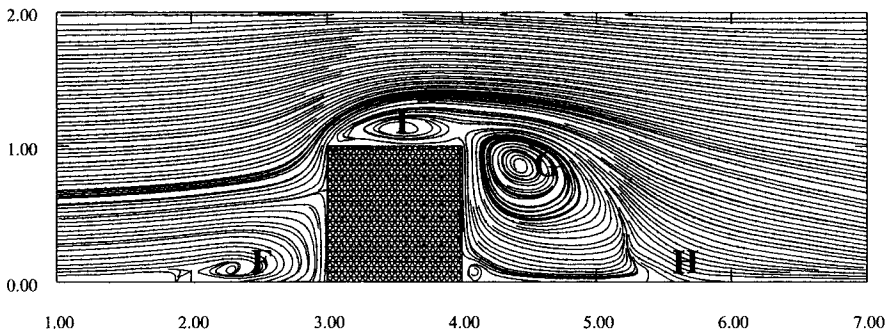
Fig. 9.6. The streamlines in the vertical center plane of the flow over a wall-mounted cube; from Shah and Ferziger (1997)

Figure 9.6 shows the streamlines of the time-averaged flow in the center plane of the flow. Many of the features described above are clearly seen including the separation zone in the upstream corner (F), which is also the head of the horseshoe vortex, the head of the arch vortex (G), the reattachment line (H), and the recirculation zone (I) above the body which does not reattach on the upper surface.

Finally, Fig. 9.7 gives a projection of the streamlines of the time-averaged flow on a plane parallel to the back face of the cube just downstream of the body. The horseshoe vortex (J) is clearly seen as are smaller corner vortices.

It is important to note that the instantaneous flow looks very different than the time-averaged flow. For example, the arch vortex does not exist in an instantaneous sense; there are vortices in the flow but they are almost always asymmetric on the two sides of the cube. Indeed, the near-symmetry of Fig. 9.5 is an indication that the averaging time is (almost) long enough.
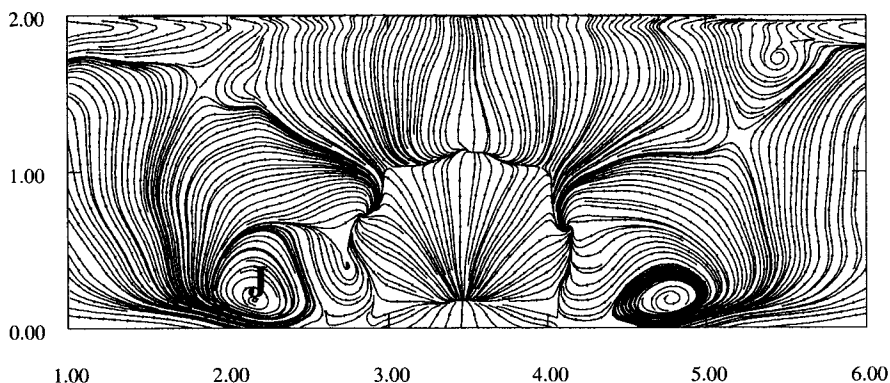
Fig. 9.7. The projection of streamlines of the flow over a wall-mounted cube onto a plane parallel to the back face, 0.1 step hight behind the cube; from Shah and Ferziger (1997)

It is clear from these results that an LES (or DNS for simpler flows) provides a great deal of information about a flow. Performance of such a simulation has more in common with doing an experiment than it does to the types of simulations described below and the qualitative information gained from it can be extremely valuable.

9.3.5 Example: Stratified Homogeneous Shear Flow

As another example of the method, we shall treat a homogeneous flow. In particular, because it displays many of the properties that are generic to flows containing both shear and stratification, stratified homogeneous shear flow is a good example of what can be done with this kind of simulation.

In homogeneous flows, the state of the turbulence is independent (in a statistical sense) of location within the flow domain. Since the properties of most of these flows change with time, the word 'statistical' is best interpreted in terms of an ensemble average. We imagine a set of flows in which all of the variables that can be controlled (energy, spectrum, boundary conditions, shear rate, stratification, etc.) are identical but the initial conditions are generated with the aid of a random number generator, each having a different seed. Thus each flow realization differs considerably in detail from the others. An ensemble average is an average over a large set of such flows. In practice, one cannot generate a very large ensemble of flows so we settle for an average over the physical domain which is permitted in homogeneous flows because every point is equivalent to every other.

By stratification, we mean that there is a density gradient in the direction of the gravity vector. If the stratification is unstable (heavy fluid over light), the instability will cause rapid mixing and the density gradient will be eliminated. We therefore consider the case of stable stratification (light fluid over

heavy). The effects of stratification are felt through a body force term in the equation for the vertical component of momentum. In many flows, density differences are small and the density may be assumed to be constant except in the body force term. This is called the Boussinesq approximation.

For a flow to be homogeneous, every point in the flow must 'feel' the same imposed strain or, more generally, the same mean velocity derivative. This means that the mean velocity must be linear in all of the coordinates. Even with this restriction, there are many possibilities. Among these are plane strain for which the mean velocity is:

$$U_1 = \Gamma x_1 \ , \quad U_2 = -\Gamma x_2 \ , \quad U_3 = 0 \ . \tag{9.16}$$

For incompressible flow, the $\partial U_i/\partial x_i = 0$ by continuity. The choice of which velocity component is set to zero is arbitrary. The case that we shall consider is pure shear for which:

$$U_1 = S x_3 \ , \quad U_2 = 0 \ , \quad U_3 = 0 \ . \tag{9.17}$$

The final example we shall give is pure rotation:

$$U_1 = \Omega x_2 \ , \quad U_2 = -\Omega x_1 \ , \quad U_3 = 0 \ . \tag{9.18}$$

It is assumed that the mean flow is imposed and maintained. Physically, this is not possible in an exact sense as the turbulence will modify the mean velocity profile. A good approximation to homogeneous shear flow is achieved in the laboratory by creating a flow with a linear velocity profile which includes a uniform velocity component. The distance down the wind tunnel divided by the mean velocity plays the role of the time. Turbulence is added to the flow by passing it through a grid. If the mean flow is maintained, one can decompose the velocity into the mean and turbulence:

$$u_i = U_i + u_i' \ . \tag{9.19}$$

Note that this is not the decomposition used in Reynolds-averaged modeling of turbulent flows. When this decomposition is substituted into the Navier-Stokes equations (including the buoyant force term) and advantage is taken of the fact that the mean flow is a solution of those equations, the result is:

$$\frac{\partial u_i'}{\partial t} + U_j \frac{\partial u_i'}{\partial x_j} + u_j' \frac{\partial U_i}{\partial x_j} + u_j' \frac{\partial u_i'}{\partial x_j} = \frac{g}{\overline{\rho}} \rho - \frac{1}{\rho} \frac{\partial p'}{\partial x_i} + \nu \frac{\partial^2 u_i'}{\partial x_j^2} \ . \tag{9.20}$$

The second term on the left hand side of this equation represents the advection of the turbulence by the mean flow. The third term is the one responsible for increasing the energy of the turbulence by vortex stretching and is often called the production term although it is more than that. In this term, the derivative $\partial U_i/\partial x_j = \Gamma_{ij}$ is constant. The fourth term represents the nonlinear interaction of turbulence with itself. The first term on the right hand side is the buoyancy term; g is the acceleration of gravity, $\overline{\rho}$ is the

mean density (assumed constant) and ρ is the deviation from that mean and includes both the mean stratification and the fluctuations.

We would like to impose periodic boundary conditions on the flow because this will permit use of spectral methods. To do this, we must be sure that the domain is sufficiently large that the velocity is not correlated across it. If this is not the case, the simulation would simply represent the behavior of a single eddy, not real turbulence. We also note that periodic conditions imply that the parts of the fluid on opposite ends of the domain are in contact. That makes sense only if they move at the same speed, which is not the case when a mean velocity field is imposed. To allow periodic conditions, it is necessary to do the simulation in a coordinate system moving with the mean velocity; the best method for achieving this was presented by Rogallo (1981) and is briefly described below.

The equations are first transformed to a coordinate system that moves with the mean flow. The terms that represent advection by the mean flow are thereby eliminated. Then the production term can be formally integrated and also eliminated. Finally, since a spectral method is used to solve the equations and the viscous term takes a simple form in Fourier space, it too can be integrated. The remaining equations are then advanced in time using a Runge-Kutta method.

For a stratified flow to be homogeneous, the mean density must also have a linear profile but, because gravity acts in only one direction, the density profile must be:

$$\rho = S_\rho x_3 + \rho' \;, \tag{9.21}$$

where the constant mean density has been removed. It can be treated in exactly the same way that the mean velocity profile was and the resulting equations can be integrated in the same way.

The presence of both stable stratification and shear means that there are two competing forces. The shear increases the intensity of the turbulence while stratification reduces it by converting some of its kinetic energy into potential energy. The interplay of the two forces is what makes this flow interesting. The parameter traditionally used to characterize their relative importance is the gradient Richardson number:

$$Ri_g = \frac{g}{\rho}\frac{\partial \rho}{\partial z} / \left(\frac{\partial U}{\partial z}\right)^2 \;, \tag{9.22}$$

which represents the relative strengths of the two forces. It can be shown that Ri_g determines whether a laminar flow is stable or not (Drazin and Reid, 1981). This parameter can also be interpreted as the ratio of the squares of the time scales associated with the two forces. The time scale associated with the stratification is the inverse of the Brunt–Väisälä frequency,

$$N = \left(\frac{g}{\rho}\frac{\partial \rho}{\partial z}\right)^{1/2} \;, \tag{9.23}$$

while the one associated with the shear is the inverse of the shear rate S:

$$S = \frac{\partial U}{\partial z} \; . \tag{9.24}$$

When the stratification is weak (low Ri_g), the shear increases the energy of the turbulence exponentially in time. If the stratification is increased while the shear rate is kept fixed (increasing Ri_g), the rate of growth of the turbulence is decreased (see Fig. 9.8). Eventually, a value of Ri_g is reached at which the turbulence neither grows nor decays ($Ri_g = 0.16$ in Fig. 9.8). At still higher stratification, the turbulence decays and eventually dies out. Before the turbulence dies, there are oscillations in the energy that represent cyclic transfer between kinetic energy and potential energy; these oscillations occur at the Brunt-Väisälä frequency.
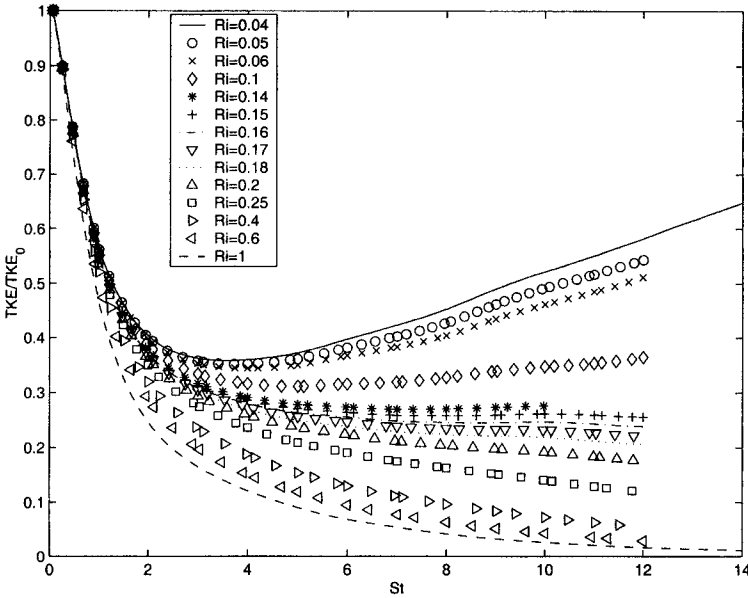
Fig. 9.8. The turbulence energy in stratified homogeneous shear flow as a function of time at various values of the gradient Richardson number

The stationary Richardson number, Ri_s (Holt et al, 1993) appears to depend only on the Reynolds number. It is important that the latter be measured at the stationary state. This is shown in Fig 9.9. It appears that, at very high Reynolds number, the Ri_s takes on a value of 0.25.

Another interesting property of these flows is that the shear rate made dimensionless with turbulence quantities, $S^* = SL/q$ (where L is the integral scale of the turbulence and $q = (2k)^{1/2}$, where k is the kinetic energy
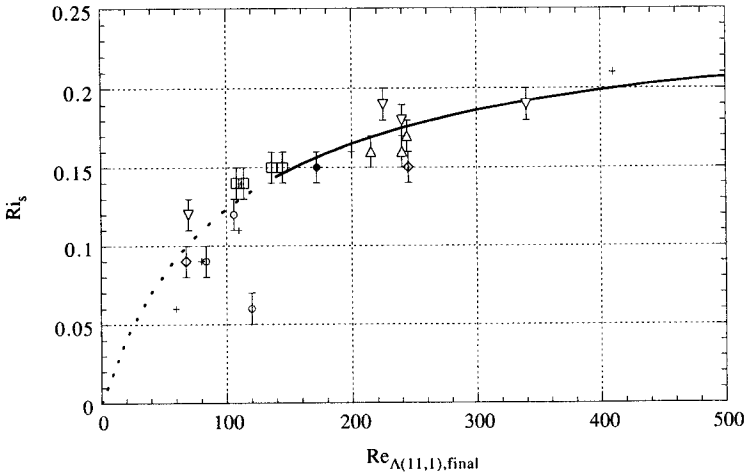
Fig. 9.9. The stationary Richardson number as a function of the Reynolds number

of the turbulence), tends to become asymptotically constant with a value of approximately 6 in homogeneous stratified sheared turbulence, at high Reynolds numbers and for Richardson numbers below the stationary value. For unstratified flow, this relation can be derived by assuming that production and dissipation are equal and the fact that, in shear flows, the structure function $b_{13} = \overline{uw}/q^2$ tends to take a constant value of approximately 0.15. These properties can be extremely useful in the construction of parameterizations for these flows.

It also turns out that the gradient Richardson number is not a particularly good parameter for describing the local state of a stratified flow. The turbulent Froude number, defined by

$$Fr_t = \frac{q}{NL} \tag{9.25}$$

performs better in this regard. It is therefore important to note that the turbulent Froude number also tends asymptotically to a constant in stationary stratified sheared homogeneous turbulence. This actually follows from the facts that S^* becomes constant and that the gradient Richardson number is constant. This was demonstrated recently by Shih et al. (2000) and should be useful in modeling. These authors also showed that the asymptotic value of Fr_t is very close to the one which maximizes the mixing efficiency, the fraction of the kinetic energy that is converted to potential energy.

9.4 RANS Models

Engineers are normally interested in knowing just a few quantitative properties of a turbulent flow, such as the average forces on a body (and, perhaps, its distribution), the degree of mixing between two incoming streams of fluid, or the amount of a substance that has reacted. Using the methods described above to compute these quantities is, to say the least, overkill. These methods should only be used as a last resort, when nothing else succeeds or, occasionally, to check the validity of a model of the type described in this section which produces less information. Because it is based on ideas proposed by Osborne Reynolds over a century ago, it is called the Reynolds-averaged method.

In Reynolds-averaged approaches to turbulence, all of the unsteadiness is averaged out i.e. all unsteadiness is regarded as part of the turbulence. On averaging, the non-linearity of the Navier-Stokes equations gives rise to terms that must be modeled, just as they did earlier. The complexity of turbulence, which was discussed briefly above, makes it unlikely that any single Reynolds-averaged model will be able to represent all turbulent flows so turbulence models should be regarded as engineering approximations rather than scientific laws.

9.4.1 Reynolds-Averaged Navier-Stokes (RANS) Equations

In a statistically steady flow, every variable can be written as the sum of a time-averaged value and a fluctuation about that value:

$$\phi(x_i, t) = \overline{\phi}(x_i) + \phi'(x_i, t) \; , \tag{9.26}$$

where

$$\overline{\phi}(x_i) = \lim_{T \to \infty} \frac{1}{T} \int_0^T \phi(x_i, t) \, \mathrm{d}t \; . \tag{9.27}$$

Here t is the time and T is the averaging interval. This interval must be large compared to the typical time scale of the fluctuations; thus, we are interested in the limit of $T \to \infty$, see Fig. 9.10. If T is large enough, $\overline{\phi}$ does not depend on the time at which the averaging is started.

If the flow is unsteady, time averaging cannot be used and it must be replaced by ensemble averaging. This concept was discussed earlier and is illustrated in Fig. 9.10:

$$\overline{\phi}(x_i, t) = \lim_{N \to \infty} \frac{1}{N} \sum_{n=1}^{N} \phi(x_i, t) \; , \tag{9.28}$$

where N is the number of members of the ensemble and must be large enough to eliminate the effects of the fluctuations. This type of averaging can be applied to any flow. We use the term *Reynolds averaging* to refer to any of
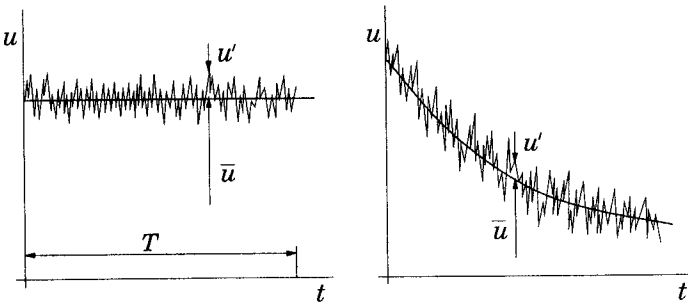
Fig. 9.10. Time averaging for a statistically steady flow (left) and ensemble averaging for an unsteady flow (right)

these averaging processes; applying it to the Navier-Stokes equations yields the Reynolds-averaged Navier-Stokes (RANS) equations.

From Eq. (9.27), it follows that $\overline{\phi'} = 0$. Thus, averaging any linear term in the conservation equations simply gives the identical term for the averaged quantity. From a quadratic nonlinear term we get two terms, the product of the average and a covariance:

$$\overline{u_i\phi} = \overline{(\bar{u}_i + u_i')(\bar{\phi} + \phi')} = \bar{u}_i\bar{\phi} + \overline{u_i'\phi'} \ . \tag{9.29}$$

The last term is zero only if the two quantities are uncorrelated; this is rarely the case in turbulent flows and, as a result, the conservation equations contain terms such as $\rho\overline{u_i'u_j'}$, called the *Reynolds stresses*, and $\rho\overline{u_i'\phi'}$, known as the *turbulent scalar flux*, among others. These cannot be represented uniquely in terms of the mean quantities.

The averaged continuity and momentum equations can, for incompressible flows without body forces, be written in tensor notation and Cartesian coordinates as:

$$\frac{\partial(\rho\bar{u}_i)}{\partial x_i} = 0 \ , \tag{9.30}$$

$$\frac{\partial(\rho\bar{u}_i)}{\partial t} + \frac{\partial}{\partial x_j}\left(\rho\bar{u}_i\bar{u}_j + \rho\overline{u_i'u_j'}\right) = -\frac{\partial\bar{p}}{\partial x_i} + \frac{\partial\bar{\tau}_{ij}}{\partial x_j} \ , \tag{9.31}$$

where the $\bar{\tau}_{ij}$ are the mean viscous stress tensor components:

$$\bar{\tau}_{ij} = \mu\left(\frac{\partial\bar{u}_i}{\partial x_j} + \frac{\partial\bar{u}_j}{\partial x_i}\right) \ . \tag{9.32}$$

Finally the equation for the mean of a scalar quantity can be written:

$$\frac{\partial(\rho\bar{\phi})}{\partial t} + \frac{\partial}{\partial x_j}\left(\rho\bar{u}_j\bar{\phi} + \rho\overline{u_j'\phi'}\right) = \frac{\partial}{\partial x_j}\left(\Gamma\frac{\partial\bar{\phi}}{\partial x_j}\right) \ . \tag{9.33}$$

The presence of the Reynolds stresses and turbulent scalar flux in the conservation equations means that the latter are not closed, that is to say, they contain more variables than there are equations. Closure requires use of some approximations, which usually take the form of prescribing the Reynolds stress tensor and turbulent scalar fluxes in terms of the mean quantities.

It is possible to derive equations for the higher order correlations e.g., for the Reynolds stress tensor, but these contain still more (and higher-order) unknown correlations that require modeling approximations. These equations will be introduced later but the important point is that it is impossible to derive a closed set of exact equations. The approximations introduced are called *turbulence models* in engineering or parametrizations in the geosciences.

9.4.2 Simple Turbulence Models and their Application

To close the equations we must introduce a turbulence model. To see what a reasonable model might be, we note, as we did in the preceding section, that in laminar flows, energy dissipation and transport of mass, momentum, and energy normal to the streamlines are mediated by the viscosity, so it is natural to assume that the effect of turbulence can be represented as an increased viscosity. This leads to the eddy-viscosity model for the Reynolds stress:

$$-\rho\overline{u_i'u_j'} = \mu_t \left(\frac{\partial \overline{u}_i}{\partial x_j} + \frac{\partial \overline{u}_j}{\partial x_i}\right) - \frac{2}{3}\rho\delta_{ij}k \ , \tag{9.34}$$

and the eddy-diffusion model for a scalar:

$$-\rho\overline{u_j'\phi'} = \Gamma_t \frac{\partial \overline{\phi}}{\partial x_j} \ . \tag{9.35}$$

In Eq. (9.34), k is the turbulent kinetic energy:

$$k = \frac{1}{2}\overline{u_i'u_i'} = \frac{1}{2}\left(\overline{u_x'u_x'} + \overline{u_y'u_y'} + \overline{u_z'u_z'}\right) \ . \tag{9.36}$$

The last term in Eq. (9.34) is required to guarantee that, when both sides of the equation are contracted (the two indices are set equal and summed over), the equation remains correct. Although the eddy-viscosity hypothesis is not correct in detail, it is easy to implement and, with careful application, can provide reasonably good results for many flows.

In the simplest description, turbulence can be characterized by two parameters: its kinetic energy, k, or a velocity, $q = \sqrt{2k}$, and a length scale, L. Dimensional analysis shows that:

$$\mu_t = C_\mu \rho q L \ , \tag{9.37}$$

where C_μ is a dimensionless constant whose value will be given later.

In the simplest practical models, mixing-length models, k is determined from the mean velocity field using the approximation $q = L\,\partial u/\partial y$ and L is a prescribed function of the coordinates. Accurate prescription of L is possible for simple flows but not for separated or highly three-dimensional flows. Mixing-length models can therefore be applied only to relatively simple flows; they are also known as zero-equation models.

The difficulty in prescribing the turbulence quantities suggests that one might use partial differential equations to compute them. Since a minimum description of turbulence requires at least a velocity scale and a length scale, a model which derives the needed quantities from two such equations is a logical choice. In almost all such models, an equation for the turbulent kinetic energy, k, determines the velocity scale. The exact equation for this quantity is not difficult to derive:

$$\frac{\partial(\rho k)}{\partial t} + \frac{\partial(\rho \bar{u}_j k)}{\partial x_j} = \frac{\partial}{\partial x_j}\left(\mu\,\frac{\partial k}{\partial x_j}\right) - \frac{\partial}{\partial x_j}\left(\frac{\rho}{2}\overline{u'_j u'_i u'_i} + \overline{p' u'_j}\right) - \rho\overline{u'_i u'_j}\frac{\partial \bar{u}_i}{\partial x_j} - \mu\overline{\frac{\partial u'_i}{\partial x_k}\frac{\partial u'_i}{\partial x_k}} \; . \tag{9.38}$$

For details of the derivation of this equation, see the book by Wilcox (1993). The terms on the left-hand side of this equation and the first term on the right-hand side need no modeling. The last term represents the product of the density ρ and the dissipation, ε, the rate at which turbulence energy is irreversibly converted into internal energy. We shall give an equation for the dissipation below.

The second term on the right-hand side represents *turbulent diffusion* of kinetic energy (which is actually transport of velocity fluctuations by the fluctuations themselves); it is almost always modeled by use of a gradient-diffusion assumption:

$$-\left(\frac{\rho}{2}\overline{u'_j u'_i u'_i} + \overline{p' u'_j}\right) \approx \frac{\mu_t}{\sigma_k}\,\frac{\partial k}{\partial x_j} \; , \tag{9.39}$$

where μ_t is the eddy viscosity defined above and σ_k is a *turbulent Prandtl number* whose value is approximately unity. In more complex models, that will not be described here, the eddy viscosity becomes a tensor.

The third term of the right-hand side of Eq. (9.38) represents the *rate of production* of turbulent kinetic energy by the mean flow, a transfer of kinetic energy from the mean flow to the turbulence. If we use the eddy-viscosity hypothesis (9.34) to estimate the Reynolds stress, it can be written:

$$P_k = -\rho\overline{u'_i u'_j}\frac{\partial \bar{u}_i}{\partial x_j} \approx \mu_t\left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}\right)\frac{\partial \bar{u}_i}{\partial x_j} \tag{9.40}$$

and, as the right hand side of this equation can be calculated from quantities that will be computed, the development of the turbulent kinetic energy equation is complete.

As mentioned above, another equation is required to determine the length scale of the turbulence. The choice is not obvious and a number of equations have been used for this purpose. The most popular one is based on the observations that the dissipation is needed in the energy equation and, in so-called equilibrium turbulent flows, i.e., ones in which the rates of production and destruction of turbulence are in near-balance, the dissipation, ε, and k and L are related by:

$$\varepsilon \approx \frac{k^{3/2}}{L} \,. \tag{9.41}$$

This idea is based on the fact, that at high Reynolds numbers, there is a cascade of energy from the largest scales to the smallest ones and that the energy transferred to the small scales is dissipated. Equation (9.41) is based on an estimate of the inertial energy transfer.

Equation (9.41) allows one to use an equation for the dissipation as a means of obtaining both ε and L. No constant is used in Eq. (9.41) because the constant can be combined with others in the complete model.

Although an exact equation for the dissipation can be derived from the Navier-Stokes equations, the modeling applied to it is so severe that it is best to regard the entire equation as a model. We shall therefore make no attempt to derive it. In its most commonly used form, this equation is:

$$\frac{\partial(\rho\varepsilon)}{\partial t} + \frac{\partial(\rho u_j \varepsilon)}{\partial x_j} = C_{\varepsilon 1} P_k \frac{\varepsilon}{k} - \rho C_{\varepsilon 2} \frac{\varepsilon^2}{k} + \frac{\partial}{\partial x_j}\left(\frac{\mu_\mathrm{t}}{\sigma_\varepsilon}\frac{\partial \varepsilon}{\partial x_j}\right) \,. \tag{9.42}$$

In this model, the eddy viscosity is expressed as:

$$\mu_\mathrm{t} = \rho C_\mu \sqrt{k} L = \rho C_\mu \frac{k^2}{\varepsilon} \,. \tag{9.43}$$

The model based on Eqs. (9.38) and (9.42) is called the k–ε model and has been widely used. This model contains five parameters; the most commonly used values for them are:

$$C_\mu = 0.09; \quad C_{\varepsilon 1} = 1.44; \quad C_{\varepsilon 2} = 1.92; \quad \sigma_k = 1.0; \quad \sigma_\varepsilon = 1.3 \,. \tag{9.44}$$

The implementation of this model in a computer code is relatively simple to carry out. The RANS equations have the same form as the laminar equations provided the molecular viscosity, μ, is replaced by the effective viscosity $\mu_\mathrm{eff} = \mu + \mu_\mathrm{t}$. The most important difference is that two new partial differential equations need to be solved. This would cause no problem but, because the time scales associated with the turbulence are much shorter than those connected with the mean flow, the equations with the k–ε model (or any

other turbulence model) are much stiffer than the laminar equations. Thus, there is little difficulty in the discretization of these equations other than one to be discussed below but the solution method has to take the increased stiffness into account.

For this reason, in the numerical solution procedure, one first performs an outer iteration of the momentum and pressure correction equations in which the value of the eddy viscosity is based on the values of k and ε at the end of the preceding iteration. After this has been completed, an outer iteration of the turbulent kinetic energy and dissipation equations is made. Since these equations are highly nonlinear, they have to be linearized prior to iteration. After completing an iteration of the turbulence model equations, we are ready to recalculate the eddy viscosity and start a new outer iteration.

The stiffness is the reason why the mean flow and turbulence equations are treated separately in the method just described; coupling the equations would make convergence very difficult to obtain. Too large a time step (or its equivalent in an iterative method) can lead to negative values of either k of ε and numerical instability. It is therefore necessary to use under-relaxation in the iterative method for these quantities; the values of the under-relaxation parameters are similar to the ones used in the momentum equations (typically 0.6–0.8).

The profiles of the turbulent kinetic energy and its dissipation are typically much more peaked near the wall than the mean velocity profile. These peaks are difficult to capture; one should probably use a finer grid for the turbulence quantities than for the mean flow but this is rarely done. If the same grid is used for all quantities, the resolution may be insufficient for the turbulence quantities and there is a chance that the solution will contain wiggles which can lead to negative values of these quantities in this region. This possibility can be avoided by locally blending the central difference scheme with a low order upwind discretization for the convective terms in the k and ε equations. This, of course, decreases the accuracy to which these quantities are calculated but is necessary if the same grid is used for all quantities.

Boundary conditions are needed for the model equations. These are generally similar to the conditions applied to any scalar equation. However, at solid walls there may be significant differences. One possibility is to solve the equations accurately right up to the wall. Then the conditions to be applied are the standard no-slip ones for the velocity. In the k–ε model, it is appropriate to set $k = 0$ at the wall but the dissipation is not zero there; instead one can use the condition:

$$\varepsilon = \nu \left(\frac{\partial^2 k}{\partial n^2} \right)_{\text{wall}} \quad \text{or} \quad \varepsilon = 2\nu \left(\frac{\partial k^{1/2}}{\partial n} \right)^2_{\text{wall}} . \tag{9.45}$$

When this is done, it is generally necessary to modify the model itself near the wall. It is argued that the effects that need to be modeled are due to the low Reynolds number of the turbulence near the wall and a number of low

Reynolds number modifications of the k–ε model have been proposed; see Patel et al. (1985) and Wilcox (1993) for a review of some of these modifications.
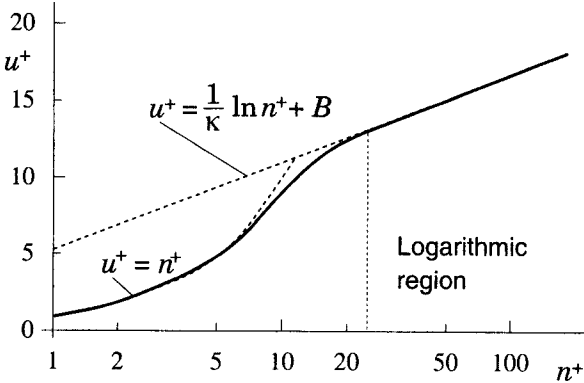
Fig. 9.11. The turbulent boundary layer: velocity profile as a function of distance normal to the wall (dashed lines are from corresponding equations, solid line represents experimental data)

At high Reynolds number, the viscous sublayer of a boundary layer is so thin that it is difficult to use enough grid points to resolve it. This problem can be avoided by using *wall functions*, which rely on the existence of a logarithmic region in the velocity profile; the velocity profile of a turbulent boundary layer is shown in Fig. 9.11. In the logarithmic layer, the profile is:

$$u^+ = \frac{\overline{v}_t}{u_\tau} = \frac{1}{\kappa} \ln n^+ + B \,, \tag{9.46}$$

where $\overline{v}_t$ is the mean velocity parallel to the wall u_τ is the shear velocity given by $u_\tau = \sqrt{|\tau_{\mathrm{w}}|/\rho}$. Here, τ_{w} is the shear stress at the wall, κ is called the von Karman constant ($\kappa = 0.41$), B is an empirical constant related to the thickness of the viscous sublayer ($B \approx 5.5$ in a boundary layer over a smooth flat plate; for rough walls, smaller values for B are obtained) and n^+ is the dimensionless distance from the wall:

$$n^+ = \frac{\rho u_\tau n}{\mu} \,. \tag{9.47}$$

It is often assumed that the flow is in local equilibrium, meaning the production and dissipation of turbulence are nearly equal. If this is the case, one can show:

$$u_\tau = C_\mu^{1/4} \sqrt{k} \,. \tag{9.48}$$

From this equation and Eq. (9.46) we can derive an expression connecting the velocity at the first grid point above the wall and the wall shear stress:

$$\tau_{\mathrm{w}} = \rho u_\tau^2 = \rho C_\mu^{1/4} \kappa \sqrt{k} \frac{\overline{v}_t}{\ln(n^+ E)} , \tag{9.49}$$

where $E = \mathrm{e}^{\kappa B}$. The control volume nearest the wall has one face that lies on the wall. In the equation for the momentum parallel to the wall for that control volume, the shear stress at the wall is required. It may be taken from Eq. (9.49) i.e. this boundary condition allows us to obtain a closed set of equations.

When these 'law of the wall' type boundary conditions are used, the diffusive flux of k through the wall is usually taken to be zero, yielding the boundary condition that the normal derivative of k is zero.

The dissipation boundary condition is derived by assuming equilibrium i.e. balance of production and dissipation in the near wall region. The production in wall region is computed from:

$$P_k \approx \tau_{\mathrm{w}} \frac{\partial \overline{v}_t}{\partial n} , \tag{9.50}$$

which is an approximation to the dominant term of Eq. (9.40) that is valid near the wall; it is valid because the shear stress is nearly constant in this region. We need the dissipation (= production) at the midpoint of the control volume closest to the wall. The velocity derivative required can be derived from the logarithmic velocity profile (9.46):

$$\left(\frac{\partial \overline{v}_t}{\partial n} \right)_{\mathrm{P}} = \frac{u_\tau}{\kappa n_{\mathrm{P}}} = \frac{C_\mu^{1/4} \sqrt{k_{\mathrm{P}}}}{\kappa n_{\mathrm{P}}} , \tag{9.51}$$

which, together with Eq. (9.49), provides a second equation relating the wall shear stress and the velocity at the first grid point. From these two equations, both quantities may be computed.

When the above approximations are used, the equation for ε is not applied in the control volume next to the wall; instead, ε is at the CV center set equal to:

$$\varepsilon_{\mathrm{P}} = \frac{C_\mu^{3/4} k_{\mathrm{P}}^{3/2}}{\kappa n_{\mathrm{P}}} . \tag{9.52}$$

This expression is derived from Eq. (9.41) using the approximation for the length scale

$$L = \frac{\kappa}{C_\mu^{3/4}} n \approx 2.5 \, n , \tag{9.53}$$

which is valid near wall under the conditions used to derive the 'law of the wall' model.

It should be noted that the above boundary conditions are valid when the first grid point is within the logarithmic region, i.e. when $n_{\mathrm{P}}^+ > 30$. Problems

arise in separated flows; within the recirculation region and, especially, in the separation and reattachment regions, the above conditions are not satisfied. Usually the possibility that wall functions may not be valid in these regions is ignored and they are applied everywhere. However, if the above conditions are violated over a large portion of the solid boundaries, serious modeling errors may result. Low Reynolds number versions of the models (or an alternative model) should be used in these regions but their accuracy has not yet been demonstrated for a wide range of flows.

At computational boundaries far from walls, the following boundary conditions can be used:

- If the surrounding flow is turbulent:

$$\overline{u}\frac{\partial k}{\partial x} = -\varepsilon \ ; \quad \overline{u}\frac{\partial \varepsilon}{\partial x} = -C_{\varepsilon 2}\frac{\varepsilon^2}{k} \ . \tag{9.54}$$

- In a free stream:

$$k \approx 0 \ ; \quad \varepsilon \approx 0 \ ; \quad \mu_{\rm t} = C_\mu \rho \frac{k^2}{\varepsilon} \approx 0 \ . \tag{9.55}$$

At an inflow boundary, k and ε are often not known; if they are available, the known values should, of course, be used. If k is not known, it is usually taken to have some small value, say $10^{-4}\,\overline{u}^2$. The value of ε should be selected so that the length scale derived from Eq. (9.41) is approximately one-tenth of the width of a shear layer or the domain size. If the Reynolds stresses and mean velocities are measured at inlet, ε can be estimated using the assumption of local equilibrium; this leads to (in a cross-section x = const.):

$$\varepsilon \approx -\overline{uv}\frac{\partial \overline{u}}{\partial y} \ . \tag{9.56}$$

A number of other *two-equation* models have been proposed; we shall describe just one of them. An obvious idea is to write a differential equation for the length scale itself; this has been tried but has not met with much success. The second most commonly used model is the k–ω model, originally introduced by Saffman but popularized by Wilcox. In this model, use is made of an equation for an inverse time scale ω; this quantity can be given various interpretations but they are not very enlightening so they are omitted here. The k–ω model uses the turbulent kinetic energy equation (9.38) but it has to be modified a bit:

$$\frac{\partial(\rho k)}{\partial t} + \frac{\partial(\rho \overline{u}_j k)}{\partial x_j} = P_k - \rho\beta^* k\omega + \frac{\partial}{\partial x_j}\left[\left(\mu + \frac{\mu_{\rm t}}{\sigma_k^*}\right)\frac{\partial \omega}{\partial x_j}\right] \ . \tag{9.57}$$

Nearly everything that was said about it above applies here. The ω equation as given by Wilcox (1998)is:

$$\frac{\partial(\rho\omega)}{\partial t} + \frac{\partial(\rho\bar{u}_j\omega)}{\partial x_j} = \alpha\frac{\omega}{k}P_k - \rho\beta\omega^2 + \frac{\partial}{\partial x_j}\left[\left(\mu + \frac{\mu_t}{\sigma_\omega^*}\right)\frac{\partial\omega}{\partial x_j}\right] . \quad (9.58)$$

In this model, the eddy viscosity is expressed as:

$$\mu_t = \rho\frac{k}{\omega} . \quad (9.59)$$

The coefficients that go into this model are a bit more complicated than those in the k–ε model. They are:

$$\alpha = \frac{5}{9} , \quad \beta = 0.075 , \quad \beta^* = 0.09 , \quad \sigma_k^* = \sigma_\omega^* = 2 , \quad \varepsilon = \beta^*\omega k . \quad (9.60)$$

The numerical behavior of this model is similar to that of the k–ε model.

The reader interested in knowing more about these models is referred to the book by Wilcox (1998). A popular variant of this model has been introduced by Menter (1993) and is used a lot in aerodynamics.

An example of the application of the k–ε model is given below.

9.4.3 The v2f Model

As should be clear from the above, a major problem with turbulence models is that the proper conditions to be applied near walls are not known. The difficulty comes from the fact that we simply do not know how some of these quantities behave near a wall. Also, the variation of the turbulent kinetic energy and, even more so, the dissipation are very rapid near a wall. This suggests that it is not a good idea to try to prescribe conditions on these quantities in that region. Another major issue is that, despite years of effort devoted to it, the development of 'low Reynolds number' models designed to treat the near-wall region, relatively little success has been achieved.

Durbin (1991) suggested that the problem is not that the Reynolds number is low near a wall (although viscous effects are certainly important). The impermeability condition (zero normal velocity) is far more important. This suggests that instead of trying to find low Reynolds number models, one should work with a quantity that becomes very small near a wall due to the impermeability condition. Such a quantity is the normal velocity (usually called v by engineers) and its fluctuations (v'^2) and so Durbin introduced an equation for this quantity. It was found that the model also required a damping function f, hence the name v^2–f (or $v2f$) model. It appears to give improved results at essentially the same cost as the k–ε model.

There are similar problems with Reynolds stress models near walls, especially with the pressure-strain terms. To remedy this problem, Durbin suggested the use of elliptic relaxation. The idea is the following. Suppose that ϕ_{ij} is some quantity that is modeled. Let the value predicted by a model be ϕ_{ij}^{m}. Instead of accepting this value as the one to be used in the model, we solve the equation:

$$\nabla^2 \phi_{ij} - \frac{1}{L^2}\phi_{ij} = \phi_{ij}^{\mathrm{m}} , \tag{9.61}$$

where L is the length scale of the turbulence, usually taken to be $L \approx k^{3/2}/\varepsilon$. The introduction of this procedure appears to relieve a great deal of the difficulty. More details on these and other similar models can be found in a recent book by Durbin and Pettersson Reif (2001).

9.4.4 Example: Flow Around an Engine Valve

We briefly present an application of the k–ε model. Valves in internal combustion engines are usually optimized by performing experiments on steady flows at several valve lifts. Lilek et al. (1991) reported the results of a combined numerical and experimental investigation of one particular geometry. The geometry was axi-symmetric, so a 2D solution method using a boundary-fitted grid was used. Second-order CDS discretization with three systematically refined grids were used; the finest had 216 × 64 CVs. By comparing the solutions on these three grids, the discretization error was estimated to be around 3% on the finest grid. Figure 9.12 shows portion of the second level grid.
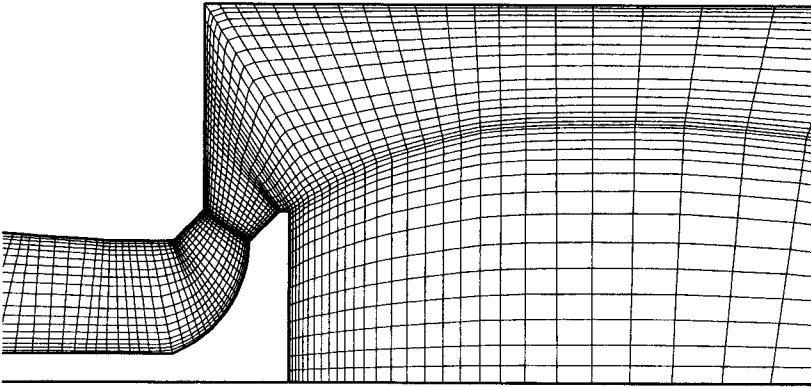
Fig. 9.12. Section of a grid (level two) used to calculate flow around a valve (from Lilek et al., 1991)

The computations were done before the experimental data was available; only the mass flow rate was prescribed. The inlet boundary was upstream of the valve, where the profiles for fully-developed annular flow (calculated separately for the same mass flow rate) were imposed. This is typical for a case in which the exact conditions at the inlet are not known. The outlet boundary was placed in the exhaust pipe, one diameter downstream of the constriction, see Fig. 9.13. Zero streamwise gradient of all variables was specified there. At the walls, the wall functions described in the preceding sections were used.

The calculated streamlines and contours of the turbulent kinetic energy are shown in Fig. 9.13. A small separation occurs at the valve throat; major recirculation regions are found behind the valve and in the corner. The high-speed flow around the valve forms an expanding annular jet which hits the cylinder wall and flows along it toward the exit, forming a wall jet. Strong turbulence is created at the edges of this jet and along the walls.
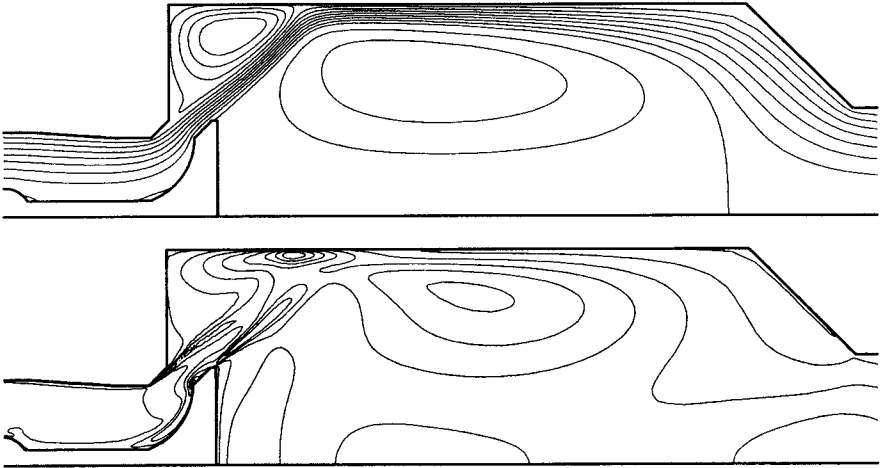
Fig. 9.13. Calculated streamlines (above) and contours of the kinetic energy (below) in flow around valve (from Lilek et al., 1991)

In Fig. 9.14 a comparison of calculated and measured axial and radial mean velocity profiles is shown. The profiles have rather complex shapes, but they are fairly well predicted; significant discrepancies between measurement and computation exist in some cross-sections and are probably due to the inadequacy of the model although this has not been definitively established.

The important question is: can such calculations be used for optimization in engineering practice? The answer is yes, if care is taken. The predictions obtained when turbulence models are used are not accurate enough that they can be accepted quantitatively without testing. However, the trends may be accurately reproduced so that the design predicted to be the best by the model also performs the best in tests. Calculations based on turbulence models can reduce the number of experimental tests required and thus reduce the cost and the time required for development of a new product. The authors know of numerous instances in which industrial corporations have used computational fluid dynamics in this way. In recent years, computation has replaced testing to a large degree and has changed the way in which experimental facilities are used.

Similar conclusions were drawn by Bertram and Jansen (1994), who used a commercial CFD code employing the k–ε turbulence model and wall functions to calculate drag of three variants of a ship hull model. They found that the computed drag coefficient was low by about 12% in absolute value; however, the relative increase or reduction of the drag when the geometry was changed was predicted with the accuracy of about 2%. The best hull form from the numerical study was also the best in the towing tank.
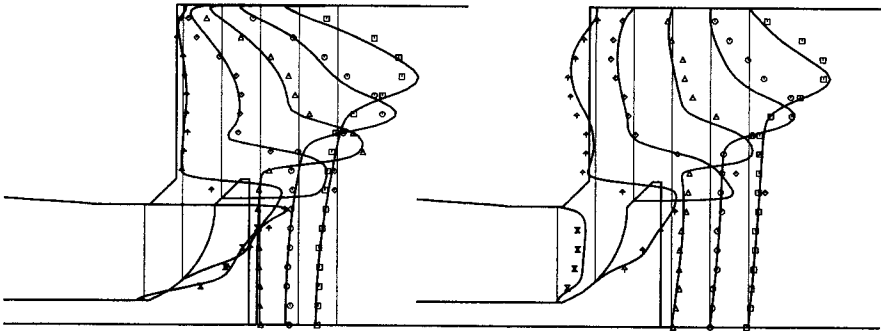
Fig. 9.14. Comparison of calculated and measured axial (left) and radial (right) velocity profiles in flow around valve (from Lilek et al., 1991)

A word of caution is necessary. New phenomena may appear in the flow when the geometry is changed and may not be well represented by the turbulence model. In such a case, computational methods may not produce accurate answers. An example is provided by a modification of the above example; Lilek et al. (1991) reported poor agreement between predicted and measured velocity profiles downstream of the valve for halved lift.

9.5 Reynolds Stress Models

Eddy-viscosity models have significant deficiencies; some are consequences of the eddy-viscosity assumption, Eq. (9.34), not being valid. In two dimensions, there is always a choice of the eddy viscosity that allows this equation to give the correct profile of the shear stress (the 1-2 component of τ_{ij}). In three-dimensional flows, the Reynolds stress and the strain rate may not be related in such a simple way. This means that the eddy viscosity may no longer be a scalar; indeed, both measurements and simulations show that it becomes a tensor quantity. Anisotropic (tensor) models based on using the k and ε equations have been proposed. They are relatively new and not yet sufficiently tested so we shall not present them here; see Craft et al. (1995) for an example. Reynolds and colleagues have developed a structure-based model that is quite promising.

The most complex models in common use today are Reynolds stress models which are based on dynamic equations for the Reynolds stress tensor $\tau_{ij} = \rho\overline{u_i' u_j'}$ itself. These equations can be derived from the Navier-Stokes equations and are:

$$\frac{\partial \tau_{ij}}{\partial t} + \frac{\partial(\overline{u}_k \tau_{ij})}{\partial x_k} = -\left(\tau_{ik}\frac{\partial \overline{u}_j}{\partial x_k} + \tau_{jk}\frac{\partial \overline{u}_i}{\partial x_k}\right) + \rho\varepsilon_{ij} - \prod_{ij} + \frac{\partial}{\partial x_k}\left(\nu\frac{\partial \tau_{ij}}{\partial x_k} + C_{ijk}\right) . \tag{9.62}$$

The first two terms of the right hand side are the production terms and require no approximation or modeling.

The other terms are:

$$\prod_{ij} = \overline{p'\left(\frac{\partial u_i'}{\partial x_j} + \frac{\partial u_j'}{\partial x_i}\right)} , \tag{9.63}$$

which is often called the pressure-strain term. It redistributes turbulent kinetic energy among the components of the Reynolds stress tensor but does not change the total kinetic energy. The next term is:

$$\rho\varepsilon_{ij} = 2\mu\overline{\frac{\partial u_i'}{\partial x_k}\frac{\partial u_j'}{\partial x_k}} , \tag{9.64}$$

which is the dissipation tensor. The last term is:

$$C_{ijk} = \rho\overline{u_i' u_j' u_k'} + \overline{p' u_i'}\delta_{jk} + \overline{p' u_j'}\delta_{ik} \tag{9.65}$$

and is often called the turbulent diffusion.

The dissipation, pressure-strain, and turbulent diffusion terms cannot be computed exactly in terms of the other terms in the equations and therefore must be modeled. The simplest and most common model for the dissipation term treats it as isotropic:

$$\varepsilon_{ij} = \frac{2}{3}\varepsilon\delta_{ij} \tag{9.66}$$

This means that an equation for the dissipation must be solved along with the Reynolds stress equations. Typically, this is taken to be the dissipation equations used in the k–ε model. More sophisticated (and therefore more complex) models have been suggested.

The simplest model for the pressure-strain term is one that assumes that the function of this term is to attempt to make the turbulence more isotropic. This model has not met with great success and a number of proposals for improvements have been made in recent years. We shall not describe or discuss these models here. The interested reader is referred to Launder (1989, 1990), Hanjalić (1994), Launder and Li (1994) and Craft and Launder (1995).

The turbulent diffusion terms are usually modeled using a gradient diffusion type of approximation. In the simplest case, the diffusivity is assumed to be isotropic and is simply a multiple of the eddy viscosity used in the models discussed earlier. In recent years, anisotropic and nonlinear models have been suggested. Again, no attempt is made to discuss them in detail here.

In three dimensions, Reynolds stress models require the solution of seven partial differential equations in addition to the equations for the mean flow. Still more equations are needed when scalar quantities need to be predicted. These equations are solved in a manner similar to that for the k–ε equations. The only additional issue is that when the Reynolds-averaged Navier–Stokes equations are solved together with a Reynolds stress model they are even stiffer than those obtained with the k–ε equations and even more care is required in their solution and the calculations usually converge more slowly.

While there is no doubt that Reynolds stress models have greater potential to represent turbulent flow phenomena more correctly than the two-equation models (see Hadžić, 1999, for some illustrative examples), their success so far has been moderate. Excellent results have been obtained for some flows in which k–ε models perform badly (e.g., swirling flows, flows with stagnation points or lines, flows with strong curvature and with separation from curved surfaces, etc.); however, in some flows their performance is hardly better at all. There is a lot of current research in this field, and new models are often proposed. Which model is best for which kind of flow (none is expected to be good for all flows) is not yet clear, partly due to the fact that in many attempts to answer this question numerical errors were too large to allow clear conclusions to be reached (Bradshaw et al., 1994). In many workshops on the subject of evaluation of turbulence models, the differences between solutions produced by different authors using supposedly the same model are often as large if not larger than the differences between the results of the same author using different models. This is one reason why numerical accuracy is emphasized in this book; its importance can not be overemphasized and constant attention to it is required.

9.6 Very Large Eddy Simulation

Researchers have attempted to build LES from the ground up starting with simple flows and going on to increasingly more complex flows in small steps. In most of these simulations, a large fraction of the energy of the turbulence was in the resolved scales and good results were achieved. Success is not assured for complex flows; this would be the case if sufficient computer resources were available, but that is not always the case.

The objective of flow simulation is usually to obtain a few selected properties of the flow at minimum cost. It is wise to use the simplest tool that will provide the desired results but it is not easy to know in advance how well each method will work. Clearly, if RANS methods are successful, there

is no reason to use LES and DNS. On the other hand, when RANS does not work, it may be a good idea to try LES.

One way to use simulation methods in the near term is to perform LES and/or DNS of 'building block' flows, ones that are structurally similar to those of actual interest. From the results, RANS models that can be applied to more complex flows can be validated and improved. RANS computations can then be the everyday tool. LES need be performed only when there are significant changes in the design.

There have been large eddy and direct numerical simulations of complex flows. Some have been spectacularly successful while others met with more limited success. It is important to determine what kinds of flows LES is good at and which ones give it problems. Large parts of the roadmap await completion.

It appears that we have to either use RANS, which is affordable, or LES, which is more accurate but rather expensive. It is natural to ask whether there is a method that provides the advantages of both RANS and LES while avoiding the disadvantages?

Flows over bluff bodies usually produce strong vortices in their wakes. These vortices produce fluctuating forces on the body in both the streamwise and spanwise directions whose prediction is very important. These include flows over buildings (wind engineering), ocean platforms, and vehicles, among others. If the vortices are sufficiently larger than the bulk of the motions that constitute the 'turbulence', it should be possible to construct a filter that retains the vortices while removing the smaller-scale motions. In so doing, one may convert an aperiodic flow into a periodic one, which may have significant consequences.

A method that accomplishes this is called either very large eddy simulation (VLES) or unsteady RANS. In this method, one uses a RANS model but computes an unsteady flow. The results often contain periodic vortex shedding. When the results of such a simulation are time-averaged, they often agree better with experiments than steady RANS computations. While there are questions about the quantitative accuracy of this approach, it certainly has some merit, at least for the near future. An example of this approach is the prediction of buoyancy-driven flows by Kenjereš (1998).

Finally, we mention a method called detached eddy simulation (DES) which has been suggested for separated flows (Travin et al., 2000). In this approach, RANS is used for the attached boundary layer and LES is applied to the free shear flow resulting from separation. This requires some means of producing the initial conditions for the LES in the separation region and this is a difficulty. Only a few simulations of this kind have been made to date and the results are not yet conclusive.

10. Compressible Flow

10.1 Introduction

Compressible flows are important in aerodynamics and turbomachinery among other applications. In high speed flows around aircraft, the Reynolds numbers are extremely high and turbulence effects are confined to thin boundary layers. The drag consists of two components, frictional drag due to the boundary layer and pressure or form drag which is essentially inviscid in nature; there may also be wave drag due to shocks which may be computed from the inviscid equations provided that care is taken to assure that the second law of thermodynamics is obeyed. If frictional drag is ignored, these flows may be computed using the inviscid momentum *Euler* equations.

Due to the importance of compressible flow in civilian and military applications, many methods of solving the equations of compressible flow have been developed. Among these are special methods for the Euler equations such as the method of characteristics and numerous methods that may be capable of extension to viscous flows. Most of these methods are specifically designed for compressible flows and become very inefficient when applied to incompressible flows. A number of variations on the reason for this can be given. One is that, in compressible flows, the continuity equation contains a time derivative which drops out in the incompressible limit. As a result, the equations become extremely stiff in the limit of weak compressibility, necessitating the use of very small time steps or implicit methods. Another version of the argument is that the compressible equations support sound waves which have a definite speed associated with them. As some information propagates at the flow velocity, the larger of the two velocities determines the allowable time step in an explicit method. In the low speed limit, one is forced to take a time step inversely proportional to the sound speed for any fluid velocity; this step size may be much smaller than the one a method designed for incompressible flows might allow.

Discretization and solution of the compressible flow equations can be carried out with methods already described. For example, to solve the time-dependent equations, one can use any of the time-advance methods discussed in Chap. 6. As the effect of diffusion is usually small in compressible flows because the Reynolds numbers are high, there may be discontinuities e.g. shocks, in the flow. Special methods for producing smooth solutions near

shocks have been constructed. These include simple upwind methods, flux blending methods, essentially non-oscillatory (ENO) methods, and total variation diminishing (TVD) methods. These will not be described here but may be found in a number of other books, e.g. Anderson et al. (1984) and Hirsch (1991).

10.2 Pressure-Correction Methods for Arbitrary Mach Number

To compute compressible flows, it is necessary to solve not only the continuity and momentum equations but also a conservation equation for the thermal energy (or one for the total energy) and an equation of state. The latter is a thermodynamic relation connecting the density, temperature, and pressure. The energy equation was given in Chap. 1; for incompressible flows it reduces to a scalar transport equation for the temperature and only the convection and heat conduction are important. In compressible flows, viscous dissipation may be a significant heat source and conversion of internal energy to kinetic energy (and vice versa) by means of flow dilatation is also important. All terms in the equations must then be retained. In integral form the energy equation is:

$$\frac{\partial}{\partial t}\int_{\Omega}\rho h\,\mathrm{d}\Omega+\int_{S}\rho h\boldsymbol{v}\cdot\boldsymbol{n}\,\mathrm{d}S=\int_{S}k\,\mathrm{grad}T\cdot\boldsymbol{n}\,\mathrm{d}S+$$
$$\int_{\Omega}\left[\boldsymbol{v}\cdot\mathrm{grad}p\;+\mathsf{S}:\mathrm{grad}\boldsymbol{v}\right]\mathrm{d}\Omega+\frac{\partial}{\partial t}\int_{\Omega}p\,\mathrm{d}\Omega\,. \tag{10.1}$$

Here h is the enthalpy per unit mass, T is the absolute temperature (K), k is the thermal conductivity and S is the viscous part of the stress tensor, $\mathsf{S}=\mathsf{T}+p\mathsf{I}$. For a perfect gas with constant specific heats, c_p and c_v, the enthalpy becomes $h=c_\mathrm{p}T$, allowing the energy equation to be written in terms of the temperature. Furthermore, under these assumptions, the equation of state is:

$$p=\rho RT\;, \tag{10.2}$$

where R is the gas constant. The set of equations is completed by adding the continuity equation:

$$\frac{\partial}{\partial t}\int_{\Omega}\rho\,\mathrm{d}\Omega+\int_{S}\rho\boldsymbol{v}\cdot\boldsymbol{n}\,\mathrm{d}S=0 \tag{10.3}$$

and the momentum equation:

$$\frac{\partial}{\partial t}\int_{\Omega}\rho\boldsymbol{v}\,\mathrm{d}\Omega+\int_{S}\rho\boldsymbol{v}\boldsymbol{v}\cdot\boldsymbol{n}\,\mathrm{d}S=\int_{S}\mathsf{T}\cdot\boldsymbol{n}\,\mathrm{d}S+\int_{\Omega}\rho\boldsymbol{b}\,\mathrm{d}\Omega\;, \tag{10.4}$$

where T is the stress tensor (including pressure terms) and $\boldsymbol{b}$ represents body forces per unit mass; see Chap. 1 for a discussion of various forms of these equations.

It is natural to use the continuity equation to compute the density and to derive the temperature from the energy equation. This leaves role of determining the pressure to the equation of state. We thus see that the roles of the various equations are quite different from the ones they play in incompressible flows. Also note that the nature of the pressure is completely different. In incompressible flows there is only the dynamic pressure whose absolute value is of no consequence; for compressible flows, it is the thermodynamic pressure whose absolute value is of critical importance.

The discretization of the equations can be carried out using the methods described in Chaps. 3 and 4. The only changes required involve the boundary conditions (which need to be different because the compressible equations are hyperbolic in character), the nature and treatment of the coupling between the density and the pressure, and the fact that shock waves, which are very thin regions of extremely large change in many of the variables, may exist in compressible flows. Below we shall extend the pressure-correction method to flows at arbitrary Mach number, following the approach of Demirdžić et al. (1993). Similar methods have been published by Issa and Lockwood (1977), Karki and Patankar (1989) and Van Doormal et al. (1987).

10.2.1 Pressure–Velocity–Density Coupling

As mentioned above, the discretization of the compressible momentum equations is essentially identical to that employed for the incompressible equations, see Chaps. 7 and 8, so we shall not repeat it here. We shall limit the discussion to the implicit pressure-correction method described in Chap. 7, but the ideas can be applied to other schemes as well.

To obtain the solution at the new time level, several outer iterations are performed; see Sect. 7.3.4 for a detailed description of the scheme for incompressible flows. If time step is small, only a few outer iterations per time step are necessary. For steady problems, the time step may be infinite and the under-relaxation parameter acts like a pseudo-time step. We consider only the segregated solution method, in which the linearized (around values from the previous outer iteration) equations for velocity components, pressure correction, temperature and other scalar variables are solved in turn. While solving for one variable, other variables are treated as known.

The discretized momentum equation for the velocity component u_i at at the mth outer iteration may be written (see Sect. 7.3.4):

$$u_{i,\mathrm{P}}^{m*} = \frac{Q_{u_i}^{m-1} - \sum_l A_l^{u_i} u_{i,l}^{m*}}{A_\mathrm{P}^{u_i}} - \frac{\Delta\Omega}{A_\mathrm{P}^{u_i}} \left(\frac{\delta p^{m-1}}{\delta x_i}\right)_\mathrm{P} . \tag{10.5}$$

Here $Q_{u_i}^{m-1}$ represents the source term minus the contribution of the pressure term; in what follows, the discretization methods applied to this term and the pressure term are not important. Also, we consider only two time level schemes and 2D geometry.

The velocities obtained by solving linearized momentum equations and using 'old' pressure and density, Eq. (10.5), do not satisfy the mass conservation equations; that's why they carry an asterisk. When the mass fluxes computed from these velocities and the 'old' density (denoted here by $\dot{m}^*$; see Eq. (8.12)) are inserted into the discretized continuity equation:

$$\frac{(\rho^{m-1} - \rho^n)\Delta\Omega}{\Delta t} + \dot{m}_\mathrm{e}^* + \dot{m}_\mathrm{w}^* + \dot{m}_\mathrm{n}^* + \dot{m}_\mathrm{s}^* = Q_\mathrm{m}^* \;, \tag{10.6}$$

there results an imbalance Q_m^* that must be eliminated by a correction method. For incompressible flows, the mass flux and the velocity are essentially equivalent and the imbalance is corrected by correcting the velocity. Since velocity correction is proportional to the gradient of the pressure correction, as was shown in Sect. 7.3.4, an equation for the pressure correction can be derived and solved. This procedure is not applicable in the compressible case.

In compressible flows the mass flux depends on both the velocity component normal to the cell face, v_n, and the (variable) density, ρ. To correct the mass flux imbalance, both the density and the velocity must be corrected. The corrected mass flux on the 'e' face of a CV can be expressed as:

$$\dot{m}_\mathrm{e}^m = (\rho^{m-1} + \rho')_\mathrm{e}(v_n^{m*} + v_n')_\mathrm{e} S_\mathrm{e} \;, \tag{10.7}$$

where ρ' and v_n' represent the density and velocity corrections, respectively. The mass flux correction is thus:

$$\dot{m}_\mathrm{e}' = (\rho^{m-1} S v_n')_\mathrm{e} + (v_n^{m*} S \rho')_\mathrm{e} + \underline{(\rho' v_n' S)_\mathrm{e}} \;. \tag{10.8}$$

The underscored term is usually neglected as it is of second order in the corrections and thus becomes zero more rapidly than the other two terms. Near convergence, this approximation is certainly permissible; one hopes that it does not affect the rate of convergence of the method when the solution is far from converged. This term can be taken into account using a predictor-corrector approach, as described in Sect. 8.8 for the treatment of non-orthogonality in the pressure-correction equation.

The first of the two remaining terms in the mass flux correction is identical to the one obtained for incompressible flows. In Sect. 8.8 it was shown that, for the colocated variable arrangement, this term can be approximated in the SIMPLE method as (see Eq. (8.59)):

$$(\rho^{m-1} S v_n')_\mathrm{e} = -(\rho^{m-1} S\, \Delta\Omega)_\mathrm{e} \overline{\left(\frac{1}{A_\mathrm{P}^{v_n}}\right)_\mathrm{e}} \left(\frac{\delta p'}{\delta n}\right)_\mathrm{e} \;, \tag{10.9}$$

where n is the coordinate in the direction of the outward normal to the cell face. Since the coefficient A_P is the same for any Cartesian velocity component, we can take $A_\mathrm{P}^{v_n} = A_\mathrm{P}^u$.

The second term in the mass flux correction, Eq. (10.8), is due to compressibility; it involves the correction to density at the CV face. If the SIMPLE method is to be extended to compressible flows, we must also express the density correction in terms of the pressure correction. This can be done as follows.

If the temperature is, for one outer iteration, regarded as fixed, we can write:

$$\rho' \approx \left(\frac{\partial \rho}{\partial p}\right)_T p' = C_\rho \, p' \, . \tag{10.10}$$

The coefficient C_ρ can be determined from the equation of state; for a perfect gas:

$$C_\rho = \left(\frac{\partial \rho}{\partial p}\right)_T = \frac{1}{RT} \, . \tag{10.11}$$

For other gases, the derivative may need to be computed numerically. The converged solution is independent of this coefficient because all corrections are then zero; only the intermediate results are affected. It is important that the connection between the density and pressure corrections be qualitatively correct and the coefficient can, of course, influence the convergence rate of the method.

The second term in the mass flux correction can now be written:

$$(v_n^{m*} S \rho')_\mathrm{e} = \left(\frac{C_\rho \, \dot{m}^*}{\rho^{m-1}}\right)_\mathrm{e} p'_\mathrm{e} \, . \tag{10.12}$$

The mass flux correction on the 'e' face of a CV is then (see Fig. 7.5):

$$\dot{m}'_\mathrm{e} = -(\rho^{m-1} S \, \Delta\Omega)_\mathrm{e} \overline{\left(\frac{1}{A_\mathrm{P}^u}\right)_\mathrm{e}} \left(\frac{\delta p'}{\delta n}\right)_\mathrm{e} + \left(\frac{C_\rho \dot{m}^*}{\rho^{m-1}}\right)_\mathrm{e} p'_\mathrm{e} \, . \tag{10.13}$$

The value of p' at the cell face center and the normal component of the gradient of p' at the cell face center need to be approximated. Any of the approximations described in Chap. 4 for convective and diffusive terms can be used for this purpose.

The continuity equation, which must be satisfied by the corrected mass fluxes and density (see Eq. (10.6)) is:

$$\frac{\rho'_\mathrm{P} \, \Delta\Omega}{\Delta t} + \dot{m}'_\mathrm{e} + \dot{m}'_\mathrm{w} + \dot{m}'_\mathrm{n} + \dot{m}'_\mathrm{s} + Q^*_\mathrm{m} = 0 \, . \tag{10.14}$$

If Eq. (10.10) is used to express ρ'_P in terms of p'_P and the approximation (10.13) of the mass flux correction is substituted into this equation, we arrive at an algebraic system of equations for the pressure correction:

$$A_\mathrm{P} p'_\mathrm{P} + \sum_l A_l\, p'_l = -Q^*_\mathrm{m} \ . \tag{10.15}$$

The coefficients in this equation depend on the approximations used for the gradients and cell face values of the pressure correction. The part which stems from the velocity correction is identical to that for the incompressible case, see Eqs. (8.59) and (10.13). The second part depends on the approximation used for the 'convective' term; it corresponds to the convective contribution to the conservation equations, see Chap. 4 and 7 for examples.

Despite the similarity in appearance to the pressure-correction equation for incompressible flows, there are important differences. The incompressible equation is a discretized Poisson equation, i.e. the coefficients represent an approximation to the Laplacian operator. In the compressible case, there are contributions that represent the fact that the equation for the pressure in a compressible flow contains convective and unsteady terms, i.e. it is actually a convected wave equation. For an incompressible flow, if the mass flux is prescribed at the boundary, the pressure may be indeterminate to within an additive constant. The presence of convective terms in the compressible pressure equation makes the solution unique.

The relative importance of the two terms in the mass flux correction depends on the type of flow. The diffusive term is of order $1/\mathrm{Ma}^2$ relative to the convective term so the Mach number is the determining factor. At low Mach numbers, the Laplacian term dominates and we recover the Poisson equation. On the other hand, at high Mach number (highly compressible flow), the convective term dominates, reflecting the hyperbolic nature of the flow. Solving the pressure-correction equation is then equivalent to solving the continuity equation for density. Thus the pressure-correction method automatically adjusts to the local nature of the flow and the same method can be applied to the entire flow region.

For the approximation of the Laplacian, central difference approximations are always applied. On the other hand, for the approximation of convective terms a variety of approximations may be used, just as is the case for the convective terms in the momentum equations. If higher-order approximations are used, the 'deferred correction' method may be used. On the left-hand side of the equation, the matrix is constructed on the basis of the first-order upwind approximation while the right-hand side contains the difference between the higher-order approximation and the first-order upwind approximation, assuring that the method converges to the solution belonging to the higher-order approximation; see Sect. 5.6 for details. Also, if the grid is severely non-orthogonal, deferred correction can be used to simplify the pressure-correction equation as described in Sect. 8.8.

These differences are reflected in the pressure-correction equation in another way. Because the equation is no longer a pure Poisson equation, the central coefficient A_P is not the negative of the sum of the neighbor coefficients. Only when $\operatorname{div}\boldsymbol{v} = 0$, is this property obtained.

10.2.2 Boundary Conditions

For incompressible flows the following boundary conditions are usually applied:

- Prescribed velocity and temperature on inflow boundaries;
- Zero gradient normal to the boundary for all scalar quantities and the velocity component parallel to the surface on a symmetry plane; zero velocity normal to such a surface;
- No-slip (zero relative velocity) conditions, zero normal stress and prescribed temperature or heat flux on a solid surface;
- Prescribed gradient (usually zero) of all quantities on an outflow surface.

These boundary conditions also hold for compressible flow and are treated in the same way as in incompressible flows. However, in compressible flow there are further boundary conditions:

- Prescribed total pressure;
- Prescribed total temperature;
- Prescribed static pressure on the outflow boundary; [1]
- At a supersonic outflow boundary, zero gradients of all quantities are usually specified.

The implementation of these boundary conditions is described below.

Prescribed Total Pressure on the Inflow Boundary. The implementation of these boundary conditions will be described for the west boundary of a two-dimensional domain with the aid of Fig. 10.1.

One possibility is to note that, for isentropic flow of an ideal gas, the total pressure is defined as:

$$p_{\mathrm{t}} = p\left(1 + \frac{\gamma - 1}{2}\,\frac{u_x^2 + u_y^2}{\gamma RT}\right)^{\frac{\gamma}{\gamma-1}} , \tag{10.16}$$

where p is the static pressure and $\gamma = c_{\mathrm{p}}/c_{\mathrm{v}}$. The flow direction must be prescribed; it is defined by:

$$\tan\beta = \frac{u_y}{u_x} , \quad \text{i.e.} \quad u_y = u_x \tan\beta . \tag{10.17}$$

[1] For incompressible flows the static pressure can also be prescribed on either the in- or outflow boundary. As the mass flux is a function of the difference in pressure between the inflow and outflow, the velocity at the inflow boundary cannot be prescribed if the pressure is prescribed at both in- and outflow boundaries.

These boundary conditions can be implemented by extrapolating the pressure from the interior of the solution domain to the boundary and then calculating the velocity there with the aid of Eqs. (10.16) and (10.17). These velocities can be treated as known within an outer iteration. The temperature can be prescribed or it can be calculated from the total temperature:

$$T_t = T\left(1 + \frac{\gamma - 1}{2}\,\frac{u_x^2 + u_y^2}{\gamma RT}\right) . \tag{10.18}$$

This treatment leads to slow convergence of the iterative method as there are many combinations of pressure and velocity that satisfy Eq. (10.16). One must implicitly take into consideration the influence of the pressure on the velocity at the inflow. One way of doing this is described below.
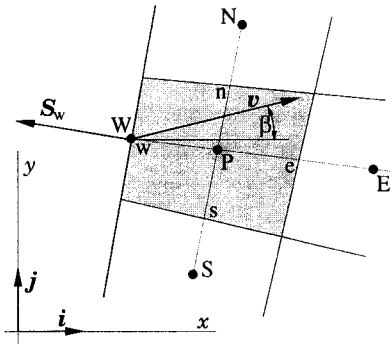
Fig. 10.1. A control volume next to an inlet boundary with a prescribed flow direction

At the beginning of an outer iteration the velocities at the inflow boundary (side 'w' in Fig. 10.1) must be computed from Eqs. (10.16) and (10.17) and the prevailing values of the pressure; they will then be treated as fixed during the outer iteration of the momentum equation. The mass fluxes at the inflow are taken from the preceding outer iteration; they should satisfy the continuity equation. From the solution of the momentum equation, (u_x^{m*}, u_y^{m*}), a new mass flux $\dot{m}^{m*}$ is computed. The 'prescribed' velocities on the inflow boundary are used to compute the mass flux there. In the following correction step, the mass flux (including its value at the inflow boundary) is corrected and mass conservation is enforced. The difference between the mass flux correction on the boundary and that at interior control volume faces is that, at the boundary, only the velocity and not the density is corrected. The velocity correction is expressed in terms of the pressure correction and not its gradient:

$$u'_{x,\mathrm{w}} = \left(\frac{\partial u_x}{\partial p}\right)_{\mathrm{w}} p'_{\mathrm{w}} = C_u p'_{\mathrm{w}} \; ; \quad u'_{y,\mathrm{w}} = u'_{x,\mathrm{w}} \tan\beta \, . \tag{10.19}$$

The coefficient C_u is determined with the aid of Eq. (10.16):

$$C_u = -\frac{\gamma R T^{m-1}}{p_\mathrm{t} u_x^{m*} \gamma \left(1+\tan^2\beta\right)\left[1+\frac{\gamma-1}{2}\,\frac{(u_x^{m*})^2(1+\tan^2\beta)}{\gamma R T^{m-1}}\right]^{\frac{1-2\gamma}{\gamma-1}}} \,. \tag{10.20}$$

The correction of the mass flux at the inflow boundary is expressed as:

$$\begin{aligned} \dot{m}'_\mathrm{w} &= [\rho^{m-1} u'_x (S^x + S^y \tan\beta)]_\mathrm{w} = \\ &[\rho^{m-1} C_u (S^x + S^y \tan\beta)]_\mathrm{w} \overline{(p')}_\mathrm{w} \,. \end{aligned} \tag{10.21}$$

The pressure correction at the boundary, $\overline{(p')}_\mathrm{w}$, is expressed by means of extrapolation from the center of the neighboring control volume i.e. as a linear combination of p'_P and p'_E. From the above equation we obtain a contribution to the coefficients A_P and A_E in the pressure correction equation for the control volume next to the boundary. Since the density is not corrected at the inflow, there is no convective contribution to the pressure correction equation there so the coefficient A_W is zero.

After solution of the pressure correction equation, the velocity components and the mass fluxes in the entire domain including the inflow boundary are corrected. The corrected mass fluxes satisfy the continuity equation within the convergence tolerance. These are used to compute the coefficients in all of the transport equations for the next outer iteration. The convective velocities at the inflow boundary are computed from Eqs. (10.16) and (10.17). The pressure adjusts itself so that the velocity satisfies the continuity equation and the boundary condition on the total pressure. The temperature at the inflow is calculated from Eq. (10.18), and the density from the equation of state (10.2).

Prescribed Static Pressure. In subsonic flows, the static pressure is usually prescribed on the outflow boundary. Then the pressure correction on this boundary is zero (this is used as a boundary condition in the pressure correction equation) but the mass flux correction is non-zero. The velocity components are obtained by extrapolation from the neighboring control volume centers, in a way similar to calculating cell-face velocities on colocated grids, e.g. for the 'e' face and mth outer iteration:

$$v_{n,\mathrm{e}}^{m*} = \overline{(v_n^{m*})}_\mathrm{e} - \Delta\Omega_\mathrm{e} \overline{\left(\frac{1}{A_\mathrm{P}^u}\right)}_\mathrm{e} \left[\left(\frac{\delta p^{m-1}}{\delta n}\right)_\mathrm{e} - \overline{\left(\frac{\delta p^{m-1}}{\delta n}\right)}_\mathrm{e}\right] , \tag{10.22}$$

where v_n is the velocity component in the direction normal to outflow boundary, which is easily obtained from Cartesian components and the known components of the unit outward normal vector, $v_n = \boldsymbol{v}\cdot\boldsymbol{n}$. The only difference from the calculation of the velocity at inner cell faces is that here the overbar denotes extrapolation from inner cells, rather than interpolation between cell centers on either side of the face. At high flow speeds, if the outflow boundary is far downstream, one can usually use the simple upwind scheme, i.e. use

the cell-center values (node P) in place of values denoted by overbar; linear extrapolation from W and P is also easily implemented on structured grids.

The mass fluxes constructed from these velocities do not, in general, satisfy the continuity equation and must therefore be corrected. Both the velocity and density need normally to be corrected, as described above. The velocity correction is:

$$v'_{n,\mathrm{e}} = -\Delta\Omega_\mathrm{e}\overline{\left(\frac{1}{A_\mathrm{P}^u}\right)_\mathrm{e}}\left(\frac{\delta p'}{\delta n}\right)_\mathrm{e} . \tag{10.23}$$

The convective (density) contribution to the mass flux correction would turn out to be zero (since $\rho'_\mathrm{e} = (C_\rho p')_\mathrm{e}$, and $p'_\mathrm{e} = 0$ since the pressure is prescribed); however, although pressure is prescribed, the temperature is not fixed (it is extrapolated from inside), so the density does need to be corrected. The simplest approximation is the first-order upwind approximation, i.e. taking $\rho'_\mathrm{e} = \rho'_\mathrm{P}$. The mass flux correction is then given by (10.13). Note, however, that the density correction is not used to correct the density at the outflow boundary – it is calculated always from the equation of state once the pressure and temperature are calculated. The mass flux, on the other hand, has to be corrected using the above expression, since only the correction used to derive the pressure-correction equation does ensure mass conservation. Since, at convergence, all corrections go to zero, the above treatment of density correction is consistent with other approximations and does not affect the accuracy of the solution, only the rate of convergence of the iterative scheme. The coefficient for the boundary node in the pressure-correction equation contains no contribution from the convective term (due to upwinding) – its contribution goes to the central coefficient A_P. The pressure derivative in the normal direction is usually approximated as:

$$\left(\frac{\delta p'}{\delta n}\right)_\mathrm{e} \approx \frac{p'_\mathrm{E} - p'_\mathrm{P}}{L_\mathrm{P,E}} , \tag{10.24}$$

where $L_\mathrm{P,E}$ is the distance from cell center P to the outflow cell face E.

The coefficient A_P in the pressure correction equation for the control volume next to the boundary thus changes compared to those at inner CVs. Due to the convective term in the pressure-correction equation and the Dirichlet boundary condition where static pressure is specified, it usually converges faster than for incompressible flow (where Neumann boundary conditions are usually applied at all boundaries and the equation is fully elliptic).

Non-Reflecting and Free-Stream Boundaries. At some portions of the boundary the exact conditions to be applied may not be known, but pressure waves and/or shocks should be able to pass through the boundary without reflection. Usually, one-dimensional theory is used to compute the velocity at boundary, based on the prescribed free-stream pressure and temperature. If the free-stream is supersonic, shocks may cross the boundary and one

makes then a distinction between the parallel velocity component, which is simply extrapolated to the boundary, and the normal component, which is computed from theory. The latter condition depends on whether compression or expansion (Prandtl-Meyer) waves hit the boundary. The pressure is usually extrapolated from the interior to the boundary, while the normal velocity component is computed using the extrapolated pressure and the prescribed free-stream Mach number.

There are many schemes designed to produce non-reflecting and free-stream boundaries. Their derivation relies on the outgoing characteristics computed via one-dimensional theory; the implementation depends on the discretization and the solution method. A detailed discussion of these (*numerical*) boundary conditions can be found in Hirsch (1991).

Supersonic Outflow. If the flow at the outflow is supersonic, all of the variables at the boundary must be obtained by extrapolation from the interior, i.e. no boundary information needs to be prescribed. The treatment of the pressure correction equation is similar to that in the case in which the static pressure is prescribed. However, since the pressure at the boundary is not prescribed but is extrapolated, the pressure correction also needs to be extrapolated – it is not zero as in the above case. Since p'_E is expressed as a linear combination of p'_P and p'_W (if the pressure gradient can be neglected, one may also set $p'_\mathrm{E} = p'_\mathrm{P}$), the node E does not occur in the algebraic equation, so $A_\mathrm{E} = 0$. The coefficients of nodes appearing in the approximation of the mass-flux correction through the boundary are different from those in the interior region.

Some examples of application of the pressure-correction scheme to solving compressible flow problems are presented below. More examples can be found in Demirdžić et al. (1993) and in Lilek (1995).

10.2.3 Examples

We present below the results of the solution of Euler equations for a flow over a circular arc bump. Figure 10.2 shows the geometry and the predicted isolines of Mach number for the subsonic, transonic and supersonic conditions. The thickness-to-chord ratio of the circular arc is 10% for subsonic and transonic cases and 4% for the supersonic case. Uniform inlet flow at Mach numbers Ma = 0.5 (subsonic), 0.675 (transonic) and 1.65 (supersonic) is specified. Since Euler equations are solved, viscosity is set to zero and slip conditions are prescribed at walls (flow tangency, as for symmetry surfaces). These problems were the test cases in a workshop in 1981 (se Rizzi and Viviand, 1981) and are often used to assess the accuracy of numerical schemes.

For subsonic flow, since the geometry is symmetric and the flow is inviscid, the flow is also symmetric. The total pressure should be constant throughout the solution domain, which is useful in assessing numerical error. In the transonic case, one shock is obtained on the lower wall. When the oncoming flow
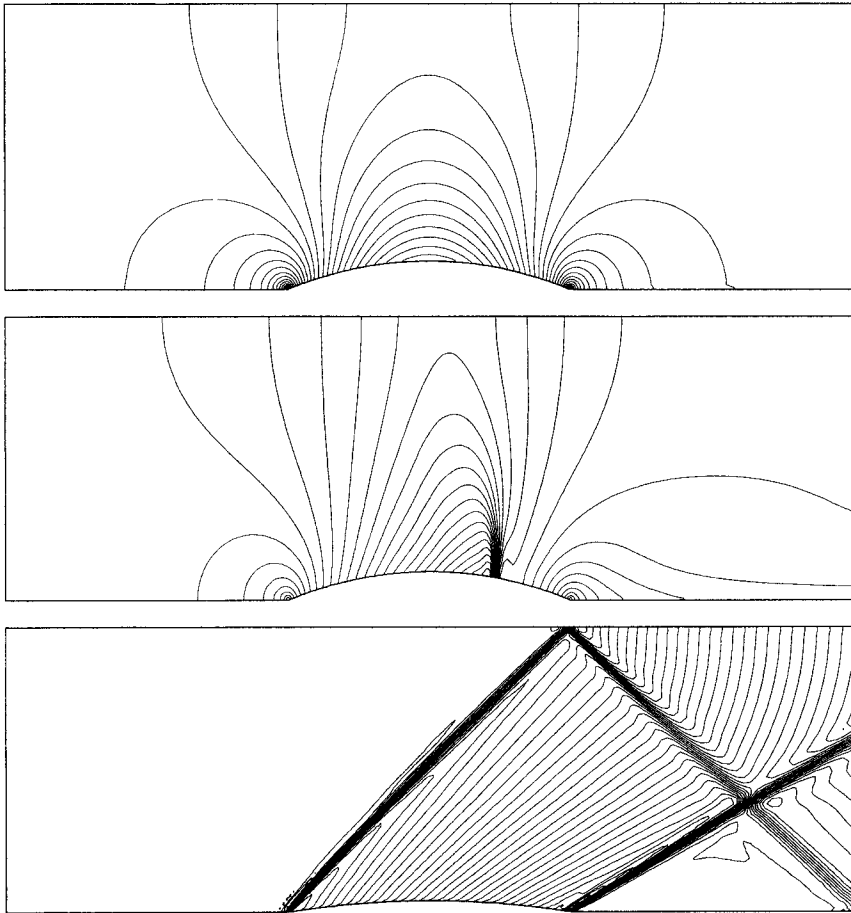
Fig. 10.2. Predicted Mach number contours for inviscid flow through a channel with a circular arc bump in lower wall: subsonic flow at $\mathrm{Ma_{in}} = 0.5$ (above), transonic flow at $\mathrm{Ma_{in}} = 0.675$ (middle), and supersonic flow at $\mathrm{Ma_{in}} = 1.65$ (bottom); from Lilek (1995)

is supersonic, a shock is generated as the flow reaches the bump. This shock is reflected by the upper wall; it crosses another shock, which issues from the end of the bump, where another sudden change in wall slope is encountered.

Figure 10.3 shows distribution of Mach number along lower and upper walls for the three cases, respectively. The solution error is very small on the finest grid and subsonic flow; this can be seen from the effects of grid refinement, as well as from the fact that the Mach numbers at both walls at the outlet are identical and equal to the inlet value. The total pressure error was below 0.25%. In the transonic and supersonic cases, grid refinement affects only the steepness of the shock; it is resolved within three grid
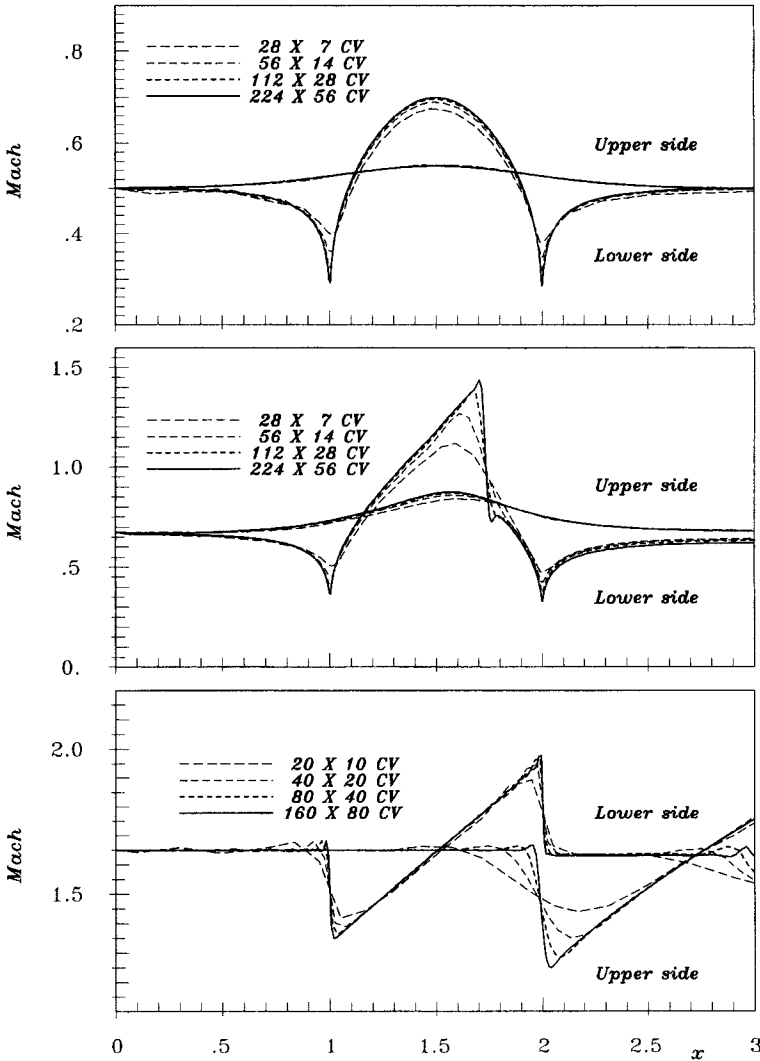
Fig. 10.3. Predicted Mach number profiles along lower and upper wall for inviscid flow through a channel with a circular arc bump in lower wall: subsonic flow at $\mathrm{Ma_{in}} = 0.5$ (above; 95% CDS, 5% UDS), transonic flow at $\mathrm{Ma_{in}} = 0.675$ (middle; 90% CDS, 10% UDS), and supersonic flow at $\mathrm{Ma_{in}} = 1.65$ (bottom; 90% CDS, 10% UDS); from Lilek (1995)

points. If central differencing is used for all terms in all equations, strong oscillations at the shocks make solution difficult. In the calculations presented here, 10% of UDS and 90% of CDS were used to reduce the oscillations; they are still present, as can be seen from Fig. 10.3, but they are limited to two grid points near the shock. It is interesting to note that the position of the

shocks does not change with grid refinement – only the steepness is improved (this was observed in many applications). The conservation properties of the FV method used and the dominant role of CDS approximations is probably responsible for this feature.

In Fig. 10.4 the Mach number contours are shown for supersonic case using pure CDS for all cell-face quantities. The coefficient A_P would be zero in this case on a uniform grid; deferred correction approach makes it possible to obtain the solution for pure CDS even in the presence of shocks and absence of diffusion terms in the equations. The solution contains more oscillations, but the shocks are better resolved.
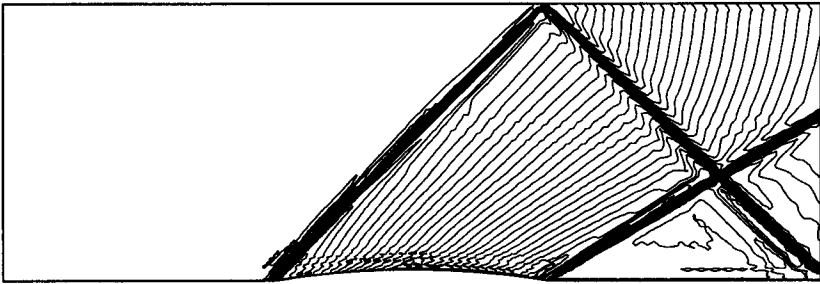
Fig. 10.4. Predicted Mach number contours for supersonic inviscid flow through a channel with a circular arc bump in lower wall (160 × 80 CV grid, 100% CDS discretization); from Lilek (1995)

Another example of the application of the pressure-correction method to high speed flow is presented below. The geometry and boundary conditions are shown in Fig. 10.5. It represents upper half of a plane, symmetric converging/diverging channel. At the inlet, the total pressure and enthalpy were specified; at the outlet, all quantities were extrapolated. The viscosity was set to zero, i.e. the Euler equations were solved. Five grids were used: the coarsest had 42 ×5 CVs, the finest 672 × 80 CVs.

The lines of constant Mach number are shown in Fig. 10.6. A shock wave is produced behind the throat, since the flow cannot accelerate due to the change in geometry. The shock wave is reflected from the walls twice before it exits through the outlet cross-section.

In Fig. 10.7 the computed pressure distribution along the channel wall is compared with experimental data of Mason et al. (1980). Results on all grids are shown. On the coarsest grid, the solution oscillates; it is fairly smooth on all other grids. As in the previous example, the locations of the shocks do not change with grid refinement but the steepness is improved as the grid is refined. The numerical error is low everywhere except near the exit, where the grid is relatively coarse; the results on the two finest grids can hardly be distinguished. Agreement with the experimental data is also quite good.
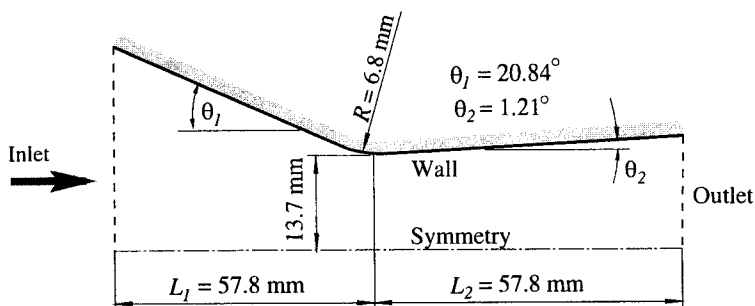
Fig. 10.5. Geometry and boundary conditions for the compressible channel flow
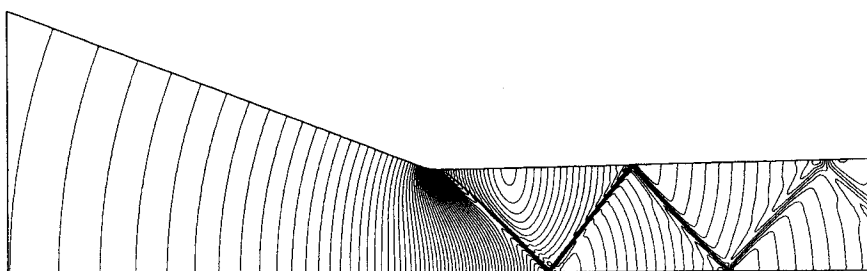
Fig. 10.6. Mach number contours in the compressible channel flow (from minimum Ma = 0.22 at inlet to maximum Ma = 1.46, step 0.02); from Lilek (1995)
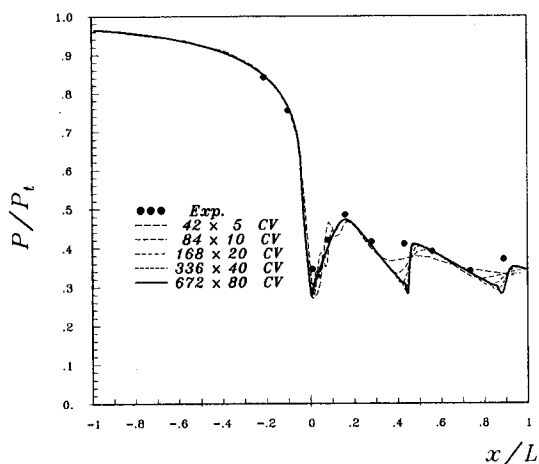
Fig. 10.7. Comparison of predicted (Lilek, 1995) and measured (Mason et al., 1980) distribution of pressure along channel wall

The solution method presented in this section tends to converge faster as the Mach number is increased (except when the CDS contribution is so large that strong oscillations appear at the shocks; in most applications it was about 90–95%). In Fig. 10.8 the convergence of the method for the solution of laminar incompressible flow at Re = 100 and for the supersonic

flow at Ma = 1.65 over a bump (see Fig. 10.4) in a channel is shown. The same grid and under-relaxation parameters are used for both flows. While in the compressible case the rate of convergence is nearly constant, in the incompressible case at low Reynolds number it gets lower as the tolerance is tightened. At very high Mach numbers, the computing time increases almost linearly with the number of grid points as the grid is refined (the exponent is about 1.1, compared to about 1.8 in case of incompressible flows). However, as we shall demonstrate in Chap. 11, the convergence of the method for elliptic problems can be substantially improved using the multigrid method, making the method very efficient. The compressible version of the method is suitable for both steady and unsteady flow problems.
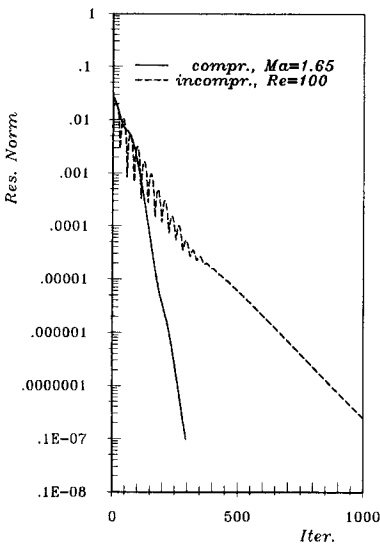
Fig. 10.8. Convergence of the pressure-correction method for laminar flow at Re = 100 and for supersonic flow at $Ma_{in} = 1.65$ over a bump in channel (160 × 80 CV grid); from Lilek (1995)

For ultimate accuracy one should apply grid refinement locally near the shocks, where the profiles suddenly change slope. The methods of applying local grid refinement and the criteria about where to refine the grid will be described in Chap. 11. Also, the blending of CDS and UDS should be applied locally, only in the vicinity of shocks, and not globally, as in the above applications. The criteria for decision where and how much of UDS to blend with CDS can be based on a monotonicity requirement on the solution, on total variation diminishing (TVD, see next section) or other suitable requirements.

10.3 Methods Designed for Compressible Flow

The method described above is an adaptation of methods designed for computing incompressible flows to the treatment of compressible flows. It was

mentioned several times in this book that there are methods specifically designed for the solution of compressible flows. In particular, these methods can be used in conjunction with the artificial compressibility methods described in Chap. 7. In this section, we briefly describe some of these methods. The purpose is to give enough information about these methods to allow comparison with the methods described above. We shall not present them in sufficient detail to allow the reader to develop codes based on them. The latter task requires a separate volume; readers interested in such a treatment are referred to the texts by Hirsch (1991) and Anderson et al. (1984).

Historically, the development of methods for the computation of compressible flows proceeded in stages. Initially (until about 1970), only the equations for linearized potential flow were solved. Later, as computer capacity increased, interest moved progressively to the non-linear potential flow equations and, in the 1980's, to the Euler equations. Methods for the viscous or Navier-Stokes equations (more properly, the RANS equations, because the high Reynolds numbers assure that the flows are turbulent) are the subject of current research and are proving very difficult to develop. So, we see that, in contrast to the situation for low speed flows, most solvers for high speed flows are designed to deal only with the inviscid case.

If there is a major theme running through these methods, it is explicit recognition that the equations are hyperbolic and thus have real characteristics along which information about the solution travels at finite speeds. The other essential issue (which arises from the existence of characteristics) is that the compressible flow equations support shock waves and other kinds of discontinuities in the solutions; the discontinuities are sharp in inviscid flows but have finite width when the viscosity is non-zero. Respecting these properties is important so it is explicitly taken into account in most methods.

These methods are mainly applied to the aerodynamics of aircraft, rockets, and turbine blades. In nearly all cases, the flow is steady. Since the speeds are high, explicit methods would need to use very small time steps and would be very inefficient. Consequently, implicit methods would be useful and have been developed. However, the nature of the equations makes it difficult to construct efficient implicit methods and, as we shall see below, many of the methods used are explicit.

The need to treat discontinuities raises another set of issues. We have seen that, in the attempt to capture any kind of rapid change in a solution, discretization methods are likely to produce results that contain oscillations or 'wiggles'. This is especially so when non-dissipative discretizations (which includes essentially all central-difference schemes) are used. A shock (or any other discontinuity) represents the extreme of a rapidly varying solution and therefore presents the ultimate challenge to discretization methods. It can be shown that no discretization method of order higher than first can guarantee a monotonic solution when the solution contains discontinuities. Since accuracy is best obtained through the use of central-difference methods (or their

equivalents in finite volume methods), many modern methods for compressible flow use central differences everywhere except near the discontinuities where special upwind methods are applied.

These are the issues that must be faced in the design of numerical methods for compressible flows. We now look, in a very general and superficial way, at some of the methods that have been proposed to deal with them.

10.3.1 An Overview of Some Specific Methods

The earliest schemes were based on explicit methods and central differencing. One of the most notable of these is the method of MacCormack (1969) which is still used. To avoid the problem of oscillations at shocks in this type of method, it is necessary to introduce artificial dissipation into the equations. The usual second-order dissipation (equivalent to ordinary viscosity) would smooth the solution everywhere so a term more sensitive to the rapid variation at the shock is needed. A fourth-order dissipative term i.e., an added term that contains fourth derivatives of the velocity, is the most common addition but higher-order terms have also been used.

The first effective implicit methods were developed by Beam and Warming (1978). Their method is based on approximate factorization of the Crank-Nicolson method and can be considered an extension of the ADI method presented in Chap. 6 to compressible flow. As with the ADI method, this method has an optimum time step for convergence to a steady solution. The use of central differences again requires addition of an explicit fourth-order dissipative term to the equations.

More recently, there has been an interest in upwind schemes of greater sophistication. The objective is always to produce a well-defined discontinuity without introducing an undue amount of error into the smooth part of the solution. One scheme for accomplishing this is the flux-vector-splitting method of Steger and Warming (1981) to which a number of modifications and extensions have been suggested. The idea is to locally split the flux (since the application is to the Euler equations, this means the convective flux of momentum) into components that flow along the various characteristics of the equations. In general, these fluxes flow in different directions. Each flux is then treated by an upwind method appropriate to the direction in which it flows. The resulting method is fairly complex but the upwinding provides stability and smoothness at discontinuities.

Finally, we mention a class of schemes that use limiters to provide smooth and accurate solutions. The earliest of these (and one of the easiest to explain) is the flux-corrected transport (FCT) method of Boris and Book (1973). In a one-dimensional version of the method, one might compute the solution using a simple first-order upwind method. The diffusive error in the solution can be estimated (one way is to use a higher-order scheme and take the difference). This estimated error is then subtracted from the solution (a so-called anti-diffusive step) but only to the extent that it does not produce oscillations.

Still more sophisticated methods are based on similar ideas and are generally referred to as *flux limiters*. The concept is to limit the flux of the conserved quantity into a control volume to a level that will not produce a local maximum or minimum of the profile of that quantity in that control volume. In total variation diminishing (TVD) schemes, one of the most popular types of these methods, the idea is to reduce the total variation of the quantity q defined by:

$$TV(q^n) = \sum_k |q_k^n - q_{k-1}^n| \,, \tag{10.25}$$

where k is a grid point index, by limiting the flux of the quantity through the control volume faces.

These methods have been demonstrated to be capable of producing very clean shocks in one-dimensional problems. The obvious way of applying them in multi-dimensional problems is to use the one-dimensional version in each direction. This is not entirely satisfactory, for reasons similar to those that make upwind methods for incompressible flows inaccurate in more than one dimension; this issue was discussed in Sect. 4.7.

TVD schemes reduce the order of approximation in the vicinity of a discontinuity. They become first order at the discontinuity itself because this is the only approximation that is guaranteed to yield a monotonic solution. The first-order nature of the scheme means that a great deal of numerical dissipation is introduced. Another class of schemes, called *essentially non-oscillatory* (ENO) schemes has been developed. They do not demand monotonicity and, instead of reducing the order of the approximation, they use different computational molecules or shape functions near a discontinuity; one-sided stencils are used to avoid interpolation across discontinuity.

In weighted ENO-schemes, several stencils are defined and checked for oscillations they produce; depending on the kind of detected oscillations, weight factors are used to define the final shape function (usually called *reconstruction polynomial*). For computational efficiency, the stencils should be few in number and compact, but to avoid oscillations while keeping a high order of approximation requires that a large number of neighbors be used in the scheme. Sophisticated methods for unstructured adaptive grids are described by Abgrall (1994), Liu et al. (1994), Sonar (1997), and Friedrich (1998), among others. These schemes are difficult to implement in implicit methods; in explicit methods they increase the computing time per time step, but the accuracy and lack of oscillations usually compensates for the higher cost.

Finally, we mention that, although they were designed for the solution of elliptic equations, multigrid methods have been applied with great success to compressible flow problems.

It will also be noted that most of the recent methods just described are explicit. This means that there are limitations on the time steps (or effective

equivalent) that can be used with them. As usual, the limitation takes the form of a Courant condition but, due to the presence of sound waves, it has the modified form:

$$\frac{|u \pm c|\, \Delta t}{\Delta x} < \alpha \,, \tag{10.26}$$

where c is the sound speed in the gas and, as usual, α is a parameter that depends on the particular time-advancement method used.

For flows that are only slightly compressible i.e., $\mathrm{Ma} = u/c \ll 1$, this condition reduces to:

$$\frac{c\, \Delta t}{\Delta x} < \alpha \,, \tag{10.27}$$

which is much more restrictive than the Courant condition:

$$\frac{u \Delta t}{\Delta x} < \alpha \tag{10.28}$$

that is usually applicable in incompressible flows. Thus methods for compressible flows tend to become very inefficient in the limit of slightly compressible flow. The pressure-correction methods presented above seem to be fairly efficient for both incompressible and compressible, steady and unsteady flows. This is why they are mostly used in general-purpose commercial codes, aimed at a wide range of applications from incompressible to highly compressible flow.

11. Efficiency and Accuracy Improvement

The best measure of the efficiency of a solution method is the computational effort required to achieve the desired accuracy. There are several methods for improving the efficiency and accuracy of CFD methods; we shall present three that are general enough to be applied to any of the solution schemes described in previous chapters.

11.1 Error Analysis and Estimation

The various types of errors which are unavoidable in the numerical solution of fluid flow problems have been briefly discussed in Sect. 2.5.7. Here we give a more detailed discussion of the various types of error and discuss how these can be estimated and eliminated. Issues of code and model validation will also be addressed.

11.1.1 Description of Errors

Modelling Errors. Fluid flow and related processes are usually described by integral or partial differential equations that represent basic conservation laws. The equations may be considered a *mathematical model* of the problem. Although the Navier-Stokes equations can be considered exact, solving them is impossible for most flows of engineering interest. Turbulence places huge demands on computer resources if it is to be simulated directly; other phenomena like combustion, multi-phase flow, chemical processes etc. are difficult to describe exactly and inevitably require the introduction of modeling approximations. Newton's and Fourier's laws are themselves only models, although they are solidly based on experimental observations for many fluids.

Even when the underlying mathematical model is nearly exact, some properties of the fluid may not be exactly known. All fluid properties depend strongly on temperature, species concentration and, possibly, pressure; this dependence is often ignored, introducing additional modeling errors (e.g. the use of the Boussinesq approximation for natural convection, the neglect of compressibility effects in low Mach-number flows, etc.).

The equations require initial and boundary conditions. These are often difficult to specify exactly. In other cases, one is forced to approximate them

for various reasons. Often what should be an infinite solution domain is taken as finite and artificial boundary conditions are applied. We often have to make assumptions about the flow at the inlet to the solution domain as well as at the lateral and outlet boundaries. Thus, even when the governing equations are exact, approximations made at the boundaries may affect the solution.

Finally, the geometry may be difficult to represent exactly; often we have to neglect details for which it is difficult to generate grids. Many codes that use structured or block-structured grids cannot be applied to very complicated problems without simplifying the geometry.

Thus, even if we were able to solve the equations and specified boundary conditions exactly, the result will not describe the flow exactly due to the errors in the model assumptions. We therefore define the *modeling error* as the difference between the real flow and the exact solution of the mathematical model.

Discretization Errors. Furthermore, we are seldom able to solve the governing equations exactly. Every numerical method produces *approximate solutions*, since various approximations have to be made to obtain an algebraic system of equations that can be solved on computer. For example, in FV methods one has to employ appropriate approximations for surface and volume integrals, variable values at intermediate locations, and time integrals. Obviously, the smaller the spatial and temporal discrete elements, the more accurate these approximations become. Using better approximations can also increase the accuracy; however, this is not a trivial matter as more accurate approximations are more difficult to program, need more computing time and storage, and may be difficult to apply to complex geometry. Usually, one selects the approximations prior to writing a code so the spatial and temporal grid resolution are the only parameters at user's disposal to control the accuracy.

The same approximation may be very accurate in one part of the flow but inaccurate elsewhere. Uniform spacing (either in space or in time) is seldom optimal, since the flow may vary strongly locally in both space and time; where the changes in variables are small, the errors will also be small. Thus, with the same number of discrete elements and the same approximations, the errors in the results may differ by an order of magnitude or more. Since the computational effort is proportional to the number of discrete elements, their proper distribution and size is essential for computational efficiency (the cost of achieving the prescribed accuracy).

We define the *discretization error* as the difference between the exact solution of the governing equations and the exact solution of the discrete approximation.

Iteration Errors. The discretization process normally produces a coupled set of non-linear algebraic equations. These are usually linearized and the linearized equations are also solved by an iterative method since direct solution is usually too expensive.

Any iteration process has to be stopped at some stage. We must therefore define a *convergence criterion* to decide when to stop the process. Usually, iteration is continued until the levels of residual has been reduced by a particular amount; this can be shown to be equivalent to reducing the error by an equal amount.

Even if the solution process is convergent and we iterate long enough, we never obtain the *exact* solution of the discretized equations; round-off errors due to finite arithmetic precision of the computer will provide a lower bound on the error. Fortunately, round-off error does not become an issue until the solution error becomes close to the arithmetic precision of computer and that is far more accuracy than is usually necessary.

We define the *iteration error* as the difference between the exact and the iterative solutions of the discretized equations. Although this kind of error has nothing to do with discretization itself, the effort required to reduce the error to a given size grows as the number of discrete elements is increased. It is therefore essential to choose an optimum level of iteration error – one that is small enough compared to the other errors (which could not be assessed otherwise) but not smaller (because the cost would be larger than necessary).

Programming and User Errors. It is often said that all computer codes have bugs – which is probably true. It is the responsibility of the code developer to try to eliminate them; an issue that we shall discuss here. It is difficult to locate programming errors by studying the code – a better approach is to devise test problems in which errors caused by bugs might show up. Results of test calculations must be carefully examined before applying the code to routine applications. One should check that the code converges at the expected rate, that the errors decrease with the number of discrete elements in the expected way, and that the solution agrees with accepted solutions produced either analytically or by another code.

A critical part of the code is the boundary conditions. The results must be checked to see if the boundary condition applied is really satisfied; it is not unusual to find that they are not. Perić (1993) discussed one such problem. Another common source of problems is the inconsistency in approximations of terms that are closely coupled; for example, in a stationary bubble the pressure drop across the free surface must be balanced by the surface tension. Simple flows for which analytical solutions are known are very useful for the verification of computer codes. For example, a code using moving grids can be examined by moving the interior grid while keeping the boundaries fixed and using stationary fluid as the initial condition; the fluid should remain stationary and should not be affected by the grid movement.

The accuracy of a solution depends not only on the discretization method and the code but also on the user of the code; it is easy to obtain bad results even with a good code! Although most user mistakes lead to errors which fall into one of the above three categories, it is important to distinguish between systematic errors, which are inherently present in the method, and

the avoidable errors, which are due to inappropriate or improper use of the code.

Many user errors are due to incorrect input data; often the error is found only after many computations have been carried out – and sometimes it is never found! Frequent errors are due to geometry scaling or parameter selection, when dimensionless form of the equations is used. Another kind of user error is due to a poor numerical grid (an inadequate distribution of grid points can increase the errors by an order of magnitude or more – or prevent one from getting solution at all).

11.1.2 Estimation of Errors

Every numerical solution contains errors; the important thing is to know how big the errors are, and whether their level is acceptable in the particular application. The acceptable level of error can vary enormously. What may be an acceptable error in an optimization study in the early design stage of a new product, where only qualitative analysis and the response of the system to design changes is important, could be catastrophic in another application.

It is thus as important to know how good the solution is for the particular application as it is to obtain the solution in the first place. Especially when using commercial codes, the user should concentrate on a careful analysis of the results and on estimation of the errors, as far as possible. This may be a great burden for a beginner, but an experienced CFD practitioner will do this routinely.

Error analysis should be done in an order reversed from the order in which they were introduced above. That is, one should begin by estimating the iteration error (which can be done within a single calculation), then the discretization error (which requires a minimum of two calculations on different grids) and, finally, the modeling error (which may require many calculations). Each of these should be an order of magnitude smaller than the one it precedes or the estimation of the later errors will not be sufficiently accurate.

Estimation of Iteration Errors. Knowing when to stop the iteration process is crucial from the point of view of computational efficiency. As a rule of thumb, the iteration errors (sometimes also called convergence errors) should be at least an order of magnitude lower than discretization errors. There is no point in iterating to the round-off level; for most engineering applications, relative accuracy (error compared to a reference value) of the three to four significant digits in any variable is more than sufficient.

There are a number of ways of estimating these errors; Ferziger and Perić (1996) analyzed three of them in detail; see also Sect. 5.7. It can be shown that the rate of reduction of error is the same as rate at which the residual and the difference between successive iterates are reduced, except in the initial stage of iteration. This was demonstrated in Fig. 7.10: the curves for the norm of

the residual, the norm of difference between successive iterates, the estimated error, and the actual iteration error are all parallel after some iterations.

Therefore, if one knows the error level at the start of computation (which is the solution itself if one starts with zero fields and somewhat lower if a rough but reasonable guess is made), then one can be confident that the error will fall 2–3 orders of magnitude if the norm of residuals (or of differences between two iterates) has fallen 3–4 orders of magnitude. This would mean that the first two or three most significant digits will not change in further iterations, and thus that the solution is accurate within 0.01–0.1 %.

A common error is to look at the magnitude of the differences between successive iterates and stop computations when they do not differ by more than a certain small number. However, the difference can be small because the iterations are slowly converging while the iteration error may be enormous. In order to estimate the magnitude of the error, one has to properly normalize the difference between successive iterates; when the convergence is slow, the normalization factor becomes large (see Sect. 5.7). On the other hand, requiring that the norm of differences fall three to four orders of magnitude is usually a safe criterion. Since the linear equation solvers in most CFD methods require the computation of residuals, the simplest practice is to monitor their norm (the sum of absolute values or square root of the sum of squares).

On a coarse grid, where the discretization errors are large, one can allow larger iteration errors; tighter tolerance is required for fine grids. This is automatically taken into account if the convergence criterion is based on the sum of residuals and not on the average residual per node, since the sum grows with growing number of nodes and thus tightens the convergence criterion.

When a new code is developed, or a new feature is added, one has to demonstrate beyond reasonable doubt that the solution process does converge until the residuals reach the round-off level. Very often, the lack of such convergence indicates that errors are present, especially in the implementation of boundary conditions. Sometimes, the limit is below the threshold at which the convergence is declared and may not be noticed. In other cases, the procedure may stop converging (or even diverge) much earlier. Once all new features have been thoroughly tested, one can return to the usual convergence criteria.

Also, if one tries to obtain a steady solution for a problem which is inherently unsteady (e.g. flow around a circular cylinder at a Reynolds number for which the von Karman vortex street is present), some iterative methods may not converge. Since each iteration can be interpreted as a pseudo-time step, it is likely that the process will not diverge, but that the residuals oscillate indefinitely. This often happens if the geometry is symmetric and the steady symmetric solution is unstable (e.g. diffusers or sudden expansions; steady solutions – both laminar and Reynolds-averaged – are asymmetric, with a larger separation region on one side). One can check whether this is

the problem by reducing the Reynolds number or computing the flow for one half of the geometry, using a symmetry boundary condition – or perform a transient computation.

Estimation of Discretization Errors. Discretization errors can only be estimated if solutions on systematically refined grids are compared; see Sect. 3.10.2 and 3.9 for more details. As noted earlier, these errors are due to the use of approximations for the various terms in the equations and the boundary conditions. For problems with smooth solutions, the quality of an approximation is described in terms of its *order*, which relates the *truncation error* of the approximation to the grid spacing to a some power; if the truncation error of a spatial derivative is proportional to say $(\Delta x)^p$, we say that the approximation is of pth order. The order is not a direct measure of the *magnitude* of the error; it indicates how the error changes when the spacing is changed. Approximations of the same order may have errors which differ as much as an order of magnitude; also, an approximation of a lower order may have a smaller error for a particular grid than one of higher order. However, as the spacing becomes smaller, the higher-order approximation will certainly become more accurate.

It is easy to find the order of many approximations using Taylor series expansion. On the other hand, different approximations may be used for different terms, so the order of the solution method as a whole may not be obvious (it is usually of the order of the least accurate approximation of a significant term in the equation). Also, errors in the implementation of the algorithm in the computer code may yield a different order than expected. It is therefore important to check the order of the method for each class of problems using the actual code.

The best way to analyze discretization errors on structured grids is to halve the spacing in each direction. However, this is not always possible; in 3D, this requires an eight-fold increase in the number of nodes. Thus, the third grid has 64 times as many points, and we may not be able to afford another refinement level. On the other hand, the errors are usually not uniformly distributed, so there is no point in refining the whole grid. Furthermore, when unstructured grids with arbitrary control volumes or elements are used, there are no local coordinate directions and the elements are refined in a different manner.

What is important is that the refinement is *substantial* and *systematic*. Increasing the number of nodes in one direction from say 54 to 62 is not very useful, except in an academic problem with uniform error distribution and a uniform grid; the refined grid should have at least 50 % more nodes in each direction than the original grid. Systematic refinement means that the grid topology and relative spatial density of grid points should remain comparable on all grid levels. A different distribution of grid points may lead to substantial changes in discretization errors without changing the number of nodes. An example is shown in Fig. 7.11: the results obtained on a non-

uniform grid, which was finer near walls, are an order of magnitude more accurate than those obtained on a uniform grid with the same number of nodes. Both solutions converge to the same grid-independent solution with the same order (second), but the errors differ in magnitude by a factor of 10 or more! Other examples with similar conclusions were shown in Figs. 6.3, 6.5, and 7.18.

The above example stresses the importance of good grid design. For practical engineering applications, grid generation is the most time-consuming task; it is often difficult to generate any grid, let alone a grid of high quality. A good grid should be as nearly orthogonal as possible (note that orthogonality has a different meaning in different methods; in a FV method, the angle between the cell-face normal and the line connecting neighboring cell centers is what counts – a tetrahedral grid may be orthogonal in this sense). It should be dense where large truncation errors are expected – hence the grid designer should know something about the solution. This is the most important criterion and is best met by using an unstructured grid with local refinement. Other criteria of quality depend on the method used (grid smoothness, aspect and expansion ratios etc.).

The simplest means of estimation of discretization errors is based on Richardson extrapolation and assumes that calculations can be done on grids sufficiently fine that monotone convergence is obtained. (If this is not the case, it is likely that the error is larger than one would like.) The method is therefore only accurate when the two finest grids are *fine enough* and the order of error reduction is known. The order may be computed from the results on three consecutive grids from the following formula provided that all three are fine enough in the above sense (see Roache, 1994, and Ferziger and Perić, 1996, for more details):

$$p = \frac{\log \left(\dfrac{\phi_{2h} - \phi_{4h}}{\phi_h - \phi_{2h}} \right)}{\log r} \,, \tag{11.1}$$

where r is the factor by which the grid density was increased ($r = 2$ if the spacing is halved), and ϕ_h denotes the solution on a grid with an average spacing h. The discretization error is then estimated as:

$$\epsilon_h \approx \frac{\phi_h - \phi_{2h}}{r^p - 1} \,. \tag{11.2}$$

Thus, when the spacing is halved, the error in the solution on one grid is equal to one third of the difference between the solutions on that and the preceding grid for a second-order method; for a first-order method, the error is equal to the aforementioned difference.

For the example from Fig. 7.11, Richardson extrapolation applied to both uniform and non-uniform grid leads to the same estimate of the grid-

independent solution within five significant digits, although the errors in the solutions are an order of magnitude different.

Note that the error can be computed for integral quantities (drag, lift, etc.) as well as for field values but the order of convergence may not be the same for all quantities. It is usually equal to the theoretical order (e.g. second) for problems with smooth solutions, e.g. laminar flows. When complicated models (for turbulence, combustion, two-phase flow etc.) or schemes with switches or limiters are used, the definition of order may be difficult. However, it is not absolutely necessary to compute quantities like the order p or what Roache (1994) calls the *grid convergence index*; it is sufficient to show the change in the computed quantity of interest for a series of grids (preferably three). If the change is monotonic and the difference decreases with grid refinement, one can easily estimate where the grid-independent solution lies. Of course, one should also use the Richardson extrapolation to estimate the grid-independent solution.

Note also that the refinement need not extend over the whole domain. If the estimate indicates that the error is much smaller in some regions than elsewhere, local refinement can be used. This is particularly true for flows around bodies, where high resolution is needed only in the vicinity of the body and in the wake. Methods using local refinement strategies are very efficient. However, care is needed; if the grid is not refined where the truncation errors are large, large errors may occur elsewhere, since the errors are subject to the same transport processes (convection and diffusion) as the variables themselves.

Estimation of Modeling Errors. Modeling errors are the most difficult ones to estimate; to do so, we need data on the real flow. In most cases, data are not available. Therefore, modeling errors are usually estimated only for some test cases, for which detailed and accurate experimental data are available, or for which accurate simulation data exist (e.g. large-eddy or direct numerical simulation data). In any case, before one can compare a computation with experiment, the iteration and discretization errors should be analyzed and shown to be small enough. In some cases the modeling and discretization errors cancel each other, so that results on a coarse grid may agree better with experimental data than ones obtained on a finer grid. Experimental data should therefore not be used to verify the code; one must use systematic analysis of the results. Only when it is proven beyond reasonable doubt that the results do converge towards a grid-independent solution and that the discretization errors are small enough, can one proceed with comparison of numerical solution and experimental data.

It is also important to bear in mind that the experimental data are only approximate, and that the measurement and data processing errors can be significant. They may also contain significant systematic errors. However, they are indispensable for the validation of models. One should compare computational results only with experimental data of high accuracy. Analysis

of the experimental data is essential if they are to be used for validation purposes.

One should also note that modeling errors differ for different quantities; for example, computed pressure drag may agree well with the measured value, but the computed friction drag may be substantially in error. Mean velocity profiles are sometimes well predicted, while the turbulence quantities may be under- or over-predicted by a factor of two. It is important to compare results with a variety of quantities in order to assure that the model really is accurate.

Detection of Programming and User Errors. A kind of errors that is difficult to quantify is the programming error. These may be simple "bugs" (typing errors that do not prevent the code from compiling) or serious algorithmic errors. The analysis of iteration and discretization errors usually helps the developer find them, but some may be so consistent that they remain undiscovered for years (if ever), especially when there are no exact reference solutions to compare.

A critical analysis of results is essential for the discovery of potential user errors; it is therefore crucial that the user have solid knowledge of fluid dynamics in general and of the problem to be solved in particular. Even if the CFD code that is being used has been validated on other flows, the user can make errors in setting-up the simulation so that the results may be significantly in error (e.g. due to errors in geometry representation, in boundary conditions, in flow parameters etc.). User errors may be difficult to spot (e.g. when an error in scaling is made and the computed flow corresponds to a different Reynolds number than anticipated); the results should therefore be critically evaluated, if possible also by someone other than the person who performed the computation.

11.1.3 Recommended Practice for CFD Uncertainty Analysis

One should distinguish between *validation* of a newly developed CFD code (or new features added to an existed code) and *validation* of an established code for a particular problem.

Validation of a CFD Code. Any new code or added feature should undergo systematic analysis with the aim of assessing the discretization errors (both spatial and temporal), of defining convergence criteria in order to assure small iteration errors, and of eliminating as many 'bugs' as possible. For this purpose one has to select a set of test cases representative of the range of problems solvable by the code, and for which sufficiently accurate solutions (analytical or numerical) are available. Since one wants to assure that the equations are correctly solved for the specified boundary conditions, experimental data are not the best way to measure the quality of numerical solutions. Reference solutions are needed to locate errors in the algorithm or

programming which may pass the tests associated with estimation of iteration and discretization errors.

One should first analyze the approximations used in the discretization to determine the order of convergence of solutions towards a grid (or time step) independent solution. This is the lowest-order truncation error in the significant terms in the equations (but note that not all terms are equally important – their importance depends on the problem). In some cases approximations of lower order may be used at a boundary than in the interior without reducing the overall order. An example is the use of one-sided first-order approximations at boundaries while second-order central differences are used in the interior; the overall convergence is second order. However, this may not be true if low-order approximations are used with Neumann-type boundary conditions.

Iteration errors should be analyzed next; as a first step, one should make a calculation in which iterations are continued until their level is reduced to the double-precision round-off level (this requires at least 12 orders of magnitude reduction of the residual). A test case, which has a known steady solution, must be selected. Otherwise, iterations may stop converging at some stage because iterations can be interpreted as pseudo-time steps and the natural instability of the flow may not allow a steady solution. An example is the case of flow around a circular cylinder around Reynolds-number 50. Once an accurate solution is available, one can compare it with solutions at intermediate stages, thus evaluating the iteration error. The error can be compared with estimates, or their reduction can be related to the reduction of the residual or the difference between successive iterates, as discussed above. This should help to establish convergence criteria (both for inner iterations, i.e. for linear equation solver, and for outer iterations, i.e. solution of the non-linear equations).

Discretization errors should be analyzed by comparing solutions on a sequence of systematically refined grids and time steps. Systematic refinement is easy for structured or block-structured grids: one creates e.g. three grids of different sizes. For unstructured grids, this task is not as straight-forward, but one can create grids with similar distributions of relative grid sizes but different absolute sizes. Systematic refinement is crucial in regions of high truncation errors, which act as sources of discretization errors, which are both convected and diffused in the same way as the dependent variables themselves. As a rule of thumb, the grid must be fine and systematically refined where the second and higher-order derivatives of the solution are large. This is typically near walls and in shear layers and wakes.

Solutions with sufficiently small iteration errors should be obtained on at least three grids and compared; both the order of convergence and the discretization error can be estimated in this way if the grids are fine enough so that monotonic convergence prevails. If this is not the case, further refinement is necessary. If the computed order is not the expected one, errors have been

made and must be sorted out. The estimated discretization error should be compared with the required accuracy.

This procedure must be repeated for a number of test cases similar to the applications in order to try to root out as many error sources as possible. Only when a systematic analysis of the results produced by the code has been made and grid and time-step independent solutions (in the sense that the discretization errors have been reliably estimated and are small enough) have been obtained, should one compare the solutions with analytical or other reference solutions. This is the final check for programming or algorithmic errors. Comparing solutions obtained on one grid with reference solutions is not meaningful, since often some quantities may accidentally agree well or some errors may cancel out.

Code validation says nothing about the accuracy with which the numerically accurate solutions represent real flows. No matter which turbulence (or other) model we use, we have to be sure that we are solving the equations that incorporate the models correctly. Comparisons of solutions obtained by different groups using the same grid and the same turbulence model but different codes often show larger differences than when one group uses the same code but different turbulence models (this is the conclusion reached at many workshops). The models appear to be differently implemented, the boundary conditions differently treated etc. This is a difficult problem for which no satisfactory solution has been found. The differences may be due to differences in implementation but, if the models used are really identical and the implementation is correct and errors have been evaluated and eliminated, every code should produce the same result and the differences should disappear. This is why we have stressed the need for validation and error evaluation.

Validation of CFD Results. Validation of CFD results includes the analysis of discretization and modeling errors; one can assume that a validated code is used with appropriate convergence criteria, so that iteration errors can be excluded.

One of the most important factors which affects the accuracy of CFD results is the quality of the numerical grid. Note that even a poor grid, if refined enough, should produce the correct solution; it will just cost more. Furthermore, even the best code may produce poor results on a bad and insufficiently refined grid, and a code based on simpler and less accurate approximations may produce excellent results if the grid is tuned for the problem being solved. (However, this is often a matter of getting the various errors to cancel each other.) Discretization errors may be reduced by a proper distribution of grid points; see Fig. 7.11.

Many commercial codes have been made sufficiently robust that they run on any grid the user might provide. However, robustness is usually achieved at the expense of accuracy (for example, by using upwind approximations). A careless user may not pay much attention to grid quality and thereby obtain inaccurate solutions with little effort. The effort invested in grid generation,

error estimation, and optimization should be related to the desired level of accuracy of the solutions. If only qualitative features of the flow are sought, a quick job may be acceptable, but for a quantitatively accurate results at reasonable cost, high grid quality is required.

Comparison with experimental data requires that the experimental uncertainty be known. It is best to compare only fully converged results (ones from which iteration and discretization error have been removed) with experimental data, because this is the only way that the effect of a model can be assessed. Experimental uncertainty bars usually extend on both sides of the reported value. If discretization errors are not small compared to the experimental uncertainty, nothing can be learned about the value of the models used. Estimation of modeling errors is the most difficult task in CFD.

In many cases the exact boundary conditions are not known and one has to make assumptions. Examples are far-field conditions for flows around bodies and inlet turbulence properties. In such a case, it is essential to vary the critical parameter (location of far-field boundary, turbulence quantities) over a substantial range to estimate the sensitivity of the solution to this factor. Often it is possible to obtain good agreement for reasonable values of the parameters but, unless the experimental data provide them, this amounts to little more than sophisticated curve fitting. That is why it is essential to choose experimental data that provide all of the necessary quantities and to discuss the importance of taking such data with the people who make the measurements. Some turbulence models are very sensitive to inlet and free-stream turbulence levels, leading to substantial changes in results for relatively minor variation in the parameters.

If a number of variants of the same geometry are to be studied, one can often rely on a validation performed for a typical representative case. It is reasonable to assume that the same grid resolution and the same model will produce discretization and modeling errors of the same order as those in the test case. While this is true in many cases, it may not always be so and care is needed. Changes in geometry may lead to the appearance of new flow phenomena (separation, secondary flow, instability etc.), which the model used may not capture. Thus, the modeling error may increase dramatically from one case to another although the change in geometry may be minor (e.g. the computation of flow around an engine valve may be accurate within 3 % for one valve opening and qualitatively wrong for a slightly smaller opening; see Lilek at al., 1991, for a more detailed description).

General Suggestions. The definition of rigid rules for validation of CFD codes and results is both difficult and impractical. While use of Richardson extrapolation is recommended wherever possible for estimating discretization errors, it may be difficult to obtain conclusive answers on all questions (e.g. the order may turn out to be different for different quantities). When several types of models are employed (for turbulence, two-phase flow, free-surface effects etc.) it may be difficult to separate different effects from one another.

The most important steps in any quantitative CFD analysis can, however, be summarized as:

- Generate a grid of appropriate structure and fineness (locally refined in regions of rapid variation of the flow and wall curvature).
- Refine the grid systematically (unstructured grids may be selectively refined: where the errors are small, refinement is not needed).
- Compute the flow on at least three grids and compare the solutions (making sure that the iteration errors are small); if the convergence is not monotonic, refine the grid again. Estimate the discretization error on the finest grid.
- If available, compare numerical solutions with reference data to estimate the modeling errors.

Any reasonable estimate of numerical errors is better than none; and numerical solutions are always *approximate solutions*, so one has to question their accuracy *all the time.*

11.2 Grid quality and optimization

Discretization errors are always reduced when a grid is refined; reliable estimation of these errors requires a grid refinement study for each new application. Optimization of a grid with a given number of grid points can reduce the discretization errors by as much (or more) than systematic refinement of a non-optimal grid. It is therefore important to pay attention to grid quality.

Grid optimization is aimed at improving the accuracy of approximations to surface and volume integrals. This depends on the discretization method used; in this section, we discuss grid features which affect the accuracy of the methods described in this book.

To obtain convective fluxes with maximum accuracy with linear interpolation and/or the midpoint rule, the line connecting two neighboring CV centers should pass through the center of the common face. In certain cases, especially when a block-structured grid is used, situations like the one shown in Fig. 11.1 are unavoidable. Most automatic grid generators create grids of this kind at protruding corners, since they usually create layers of hexahedra or prisms at boundaries. To improve the accuracy without adaptation one should locally refine the grid as shown in Fig. 11.1. This reduces the distance between cell-face center k and the point at which the straight line connecting nodes C and N_k passes through the cell face, k'. The distance between these two points, relative to the size of the cell face (e.g. $\sqrt{S_k}$) is a measure of the grid quality. Cells in which this distance is too large should be refined until the distance between k' and k is reduced to an acceptable level.

Maximum accuracy for the diffusive flux is obtained when the line connecting the neighboring CV centers is orthogonal to the cell face and passes through the cell-face center. Orthogonality increases the accuracy of the
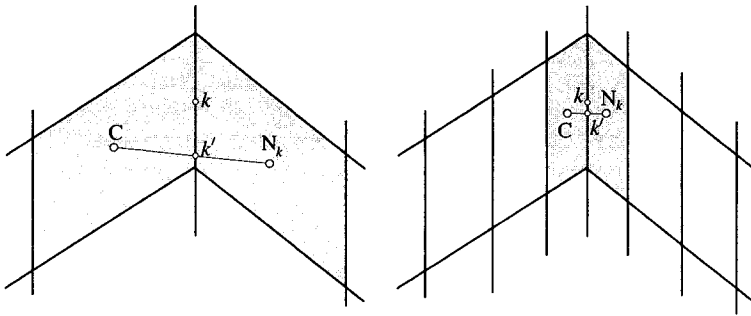
Fig. 11.1. An example of poor grid quality due to a large distance between k and k' (left) and the improvement through local grid refinement (right)

central-difference approximation to the derivative in the direction of cell-face normal:

$$\left(\frac{\partial \phi}{\partial n}\right)_{k'} \approx \frac{\phi_{N_k} - \phi_{C}}{|\boldsymbol{r}_{N_k} - \boldsymbol{r}_{C}|} . \tag{11.3}$$

This approximation is second-order accurate at the midpoint between the two cell centers; higher-order approximations can be obtained using polynomial fits even when k' is not midway between the nodes. If the non-orthogonality is not negligible, estimation of the normal derivative requires the use of many nodes. This may lead to convergence problems.

If k' is not the cell-face center, the assumption that the value at k' represents the mean value over the cell face is no longer second-order accurate. Although corrections or alternative approximations are possible, most general-purpose CFD codes use simple approximations such as (11.3) and the accuracy is substantially reduced if the grid properties are unfavorable. A simple correction which restores the second-order accuracy is:

$$\phi_k = \phi_{k'} + \overline{(\text{grad}\,\phi)}_{k'} \cdot (\boldsymbol{r}_k - \boldsymbol{r}_{k'}) . \tag{11.4}$$

In most finite-volume methods it is not important that the grid lines be orthogonal at CV corners; only the angle between the line connecting neighboring CV centers and the cell-face normal matters (see angle θ in Fig. 11.2). A tetrahedral grid can be orthogonal in this sense. An angle θ that is far from 90° can lead to large errors and convergence problems and should be avoided. In the situation shown in Fig. 11.1, the line connecting the neighboring CV centers is orthogonal to the cell face, so that the gradient at k' is accurately computed but, due to the large distance between k' and k, the accuracy of the flux integrated over the surface is poor.

Other kinds of undesirable distortions of CVs may be encountered. Two are depicted in Fig. 11.3. In one case, the upper face of a regular hexahedral CV is rotated around its normal, warping the adjacent faces. In the other
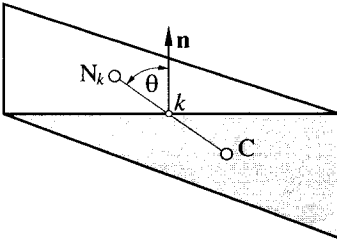
Fig. 11.2. An example of grid non-orthogonality

case, the top face is sheared in its own plane. Both features are undesirable and should be avoided if at all possible.
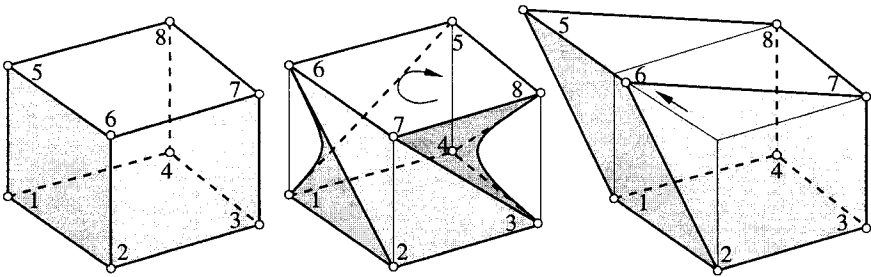
Fig. 11.3. An example of poor grid quality due to warping (middle) and distortion (right)

Grids made of triangles in 2D and tetrahedra in 3D can cause unexpected problems. One such situation is shown in Fig. 11.4; the CV centered around node C is very narrow. While the velocity component in x-direction at node C is strongly coupled to the pressure at nodes N_2 and N_3, the pressure gradient in y-direction must be computed from much more widely spaced nodes. As a result, the y-component of the velocity vector at node C may attain large values and this component may oscillate. The problem is even more pronounced in 3D. The pressure-velocity coupling algorithm may not be able to remove these oscillations and the outer iterations may not converge. It is therefore important that triangles or tetrahedra with large aspect ratios be avoided. CVs of this kind may be produced near solid walls if one tries to resolve the boundary layer by reducing the distance between grid nodes in the wall-normal direction, see Fig. 11.4. Using layers of hexahedra or prisms near walls usually reduces the problem considerably.

If the computational nodes are placed at the CV centroids, volume integrals approximated by the midpoint rule are second-order accurate. However, CVs may sometimes be so deformed, that the centroid is actually situated outside the CV. This should be avoided. The grid generator should inspect
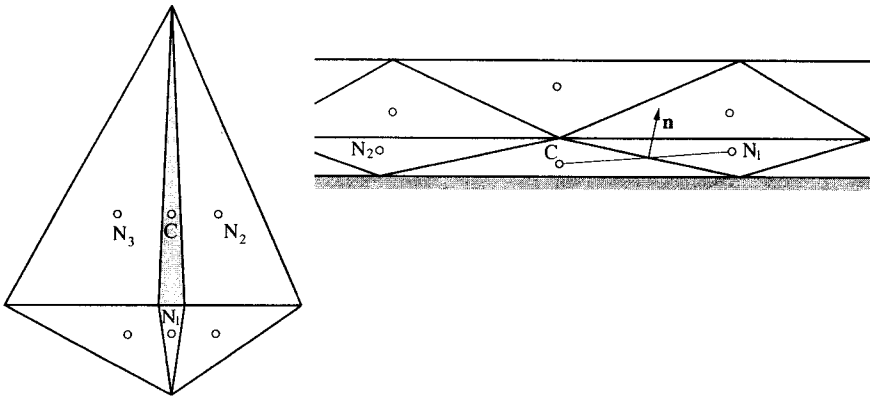
Fig. 11.4. An example of poor-quality triangular and tetrahedral grids.

the grid it produces and indicate to the user that problematic cells exist unless it is able to correct them automatically.

Some of these problems can be avoided by subdividing problematic cells (and, possibly, some surrounding cells). Unfortunately, in some cases the only solution is the generation of a new grid.

11.3 Multigrid Methods for Flow Calculation

Almost all iterative solution methods converge more slowly on finer grids. The rate of convergence depends on the method; for many methods, the number of outer iterations to obtain a converged solution is linearly proportional to the number of nodes in one coordinate direction. This behavior is related to the fact that information travels only one grid per iteration and, for convergence, information has to travel back and forth across the domain several times. Multigrid methods, for which the number of iterations is independent of the number of grid points, have received a lot of attention in the past decade. It has been demonstrated by many authors, including the present ones, that solution of the Navier-Stokes equations by multigrid methods is very efficient. Experience with a wide variety of laminar and turbulent flows shows a tremendous reduction in computing effort resulting from implementation of the multigrid idea (see the review paper by Wesseling, 1990). We give here a brief summary of a version of the method used by the authors; many other variants are possible, see proceedings of the international conferences devoted to multigrid methods in CFD, e.g. McCormick (1987), and Hackbusch and Trottenberg (1991).

In Chap. 5 we presented a multigrid method for solving linear systems of equations efficiently. We saw there that the multigrid method uses a hierarchy of grids; in the simplest case, the coarse ones are subsets of the fine ones. It

is ideal for solving the Poisson-like pressure or pressure-correction equation when fractional step or other explicit time-stepping methods are applied to unsteady flows because accurate solution of the pressure equation is required; this is often done in LES and DNS of flows in complex geometry. On the other hand, when implicit methods are used, the linear equations need not be solved very accurately at each iteration; reduction of the residual level by one order of magnitude suffices and can usually be achieved with a few iterations of one of the basic solvers such as ILU or CG. More accurate solution will not reduce the number of outer iterations but may increase the computing time.

For steady flow problems, we have seen that implicit solution methods are preferred and acceleration of the outer iterations is very important. Fortunately, the multigrid method can be applied to outer iterations. The sequence of operations that constitute one outer iteration is then considered as the 'smoother' in accord with multigrid terminology.

In a multigrid version of a finite volume method for steady flows on a structured grid, each coarse grid CV is composed of four CVs of the next finer grid in 2D and eight in 3D. The coarsest grid is usually generated first and the solution process starts by solving the problem on it. Each CV is then subdivided in finer CVs. After a converged solution is found on the coarsest grid, it is interpolated to the next finer grid to provide the starting solution. Then a two-grid procedure is begun. The process is repeated until the finest grid is reached and a solution on it is obtained. As noted earlier, this strategy is called *full multigrid* procedure (FMG).

After m outer iterations on the grid with spacing h, the short-wavelength error components have been removed and the intermediate solution satisfies the following equation:

$$A_h^m \phi_h^m - Q_h^m = \rho_h^m \, , \tag{11.5}$$

where ρ_h^m is the residual vector after the mth iteration. The solution process is now transferred to the next coarser grid whose spacing is $2h$. As noted earlier, both the cost of an iteration and the convergence rate are much more favorable on the coarse grid, giving the method its efficiency.

The equations solved on the coarse grid should be smoothed versions of the fine grid equations. With a careful choice of definitions, one can assure that the equations solved appear identical to the ones solved earlier on that grid i.e. the coefficient matrix is the same. However, the equations now contain an additional source term:

$$\hat{A}_{2h}\hat{\phi}_{2h} - \hat{Q}_{2h} = \underline{\tilde{A}_{2h}\tilde{\phi}_{2h} - \tilde{Q}_{2h} - \tilde{\rho}_{2h}} \, . \tag{11.6}$$

If set to zero, the left-hand side of Eq. (11.6) would represent the coarse grid equations. The right-hand side contains the correction that assures that the solution is a smoothed fine grid solution rather than the coarse grid solution itself. The additional terms are obtained by smoothing ('restricting') of the fine grid solution and residual; they remain constant during the iterations on the coarse grid. The initial values of all terms on the left hand side of the

above equation are the corresponding terms on the right hand side. If the fine-grid residual is zero, the solution will be $\hat{\phi}_{2h} = \tilde{\phi}_{2h}$.

Only if the residual on the fine grid is non-zero, will the coarse-grid approximation change from its initial value (since the problem is non-linear, the coefficient matrix and the source term also change, which is why these terms carry a ˆ symbol). Once the solution on the coarse grid is obtained (within a certain tolerance), the correction

$$\phi' = \hat{\phi}_{2h} - \tilde{\phi}_{2h} \tag{11.7}$$

is transferred by interpolation ('prolonged') to the fine grid and added to the existing solution ϕ_h^m. With this correction, much of the low-frequency error in the solution on the fine grid is removed, saving a lot of iterations on the fine grid. This process is continued until the solution on fine grid is converged. Richardson extrapolation may then be used to obtain an improved starting guess for the next finer grid, and a three level V-cycle is initiated, and so on.

For structured grids, simple bilinear (in 2D) or trilinear (in 3D) interpolation is usually used to transfer variable values from fine to coarse grids and corrections from coarse to fine grids. Although more complex interpolation techniques can and have been used, in most cases this simple technique is quite adequate.
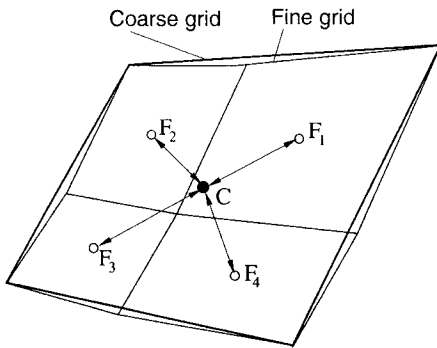
Fig. 11.5. Transfer of variables from fine to coarse grid and vice-versa

Another way to transfer a variable from one grid to another is to compute the gradient of that variable at the CV centers of the grid on which it was calculated (coarse or fine). An efficient way to calculate gradients at the centers of arbitrary CVs using Gauss theorem was described in Chap. 8. It is then easy to calculate the variable value anywhere nearby using this gradient (this corresponds to linear interpolation). For the case shown in Fig. 11.5, we can calculate the coarse-grid variable value at node C by averaging the values calculated using the fine-grid CV gradients:

$$\phi_{\mathrm{C}} = \frac{1}{N_{\mathrm{f}}} \sum_{i=1}^{N_{\mathrm{f}}} [\, \phi_{\mathrm{F}_i} + (\mathrm{grad}\phi)_{\mathrm{F}_i} \cdot (\boldsymbol{r}_{\mathrm{C}} - \boldsymbol{r}_{\mathrm{F}_i}) \,] \,, \tag{11.8}$$

where N_{f} is the number of fine-grid CVs in one coarse-grid CV (on structured grids, four in 2D and eight in 3D). The coarse CV does not need to know which fine CVs belong to it – it only needs to know how many there are. On the other hand, each fine-grid CV (child) knows which coarse grid CV (parent) it belongs to; it has only one parent.

Similarly, the coarse grid correction is easily transferred to the fine grid. One calculates the gradient of the correction at the coarse grid CV center; the correction at the fine-grid nodes which lie within this CV are calculated from:

$$\phi'_{\mathrm{F}_i} = \phi'_{\mathrm{C}} + (\mathrm{grad}\phi')_{\mathrm{C}} \cdot (\boldsymbol{r}_{\mathrm{F}_i} - \boldsymbol{r}_{\mathrm{C}}) \,. \tag{11.9}$$

This interpolation is more accurate than simple injection of the coarse CV correction to all fine CVs within it (this can also be done, but smoothing of the correction is necessary after prolongation). On structured grids, one can easily implement other kinds of polynomial interpolation.

In FV methods, the conservation property can be used to transfer the mass fluxes and residuals from the fine to the coarse grid. In 2D, the coarse grid CV is made up of four fine grid CVs and the coarse grid equation should be the sum of its daughter fine grid CVs equations. The residuals are thus simply summed over fine grid CVs, and the initial mass flux at coarse CV faces is the sum of the mass fluxes at the fine CV faces. During calculations on coarse grids, the mass fluxes are not calculated using restricted velocities but are *corrected* using velocity corrections, (the former would be less accurate and the correction is smoother than the variable itself). For the generic variables we may write:

$$\tilde{\phi}_{k-1} = I_k^{k-1} \phi_k^m \quad \text{and} \quad \phi_k^{m+1} = \phi_k^m + I_{k-1}^k \phi'_{k-1} \,, \tag{11.10}$$

where I_k^{k-1} is the operator describing the transfer from fine to coarse grid and I_{k-1}^k is the operator describing the transfer from coarse to fine grid; Eqs. (11.8) and (11.9) provide examples of these operators.

The treatment of the pressure terms in the momentum equations deserves special mention. Since initially $\hat{p} = \tilde{p}$ and the pressure terms are linear, we may work with the difference $p' = \hat{p} - \tilde{p}$. Then we do not need to restrict the pressure from fine to coarse grids. Note that this is not the pressure correction p' of SIMPLE and related algorithms; it is a correction of the finer grid pressure and is based on the velocity corrections $u'_i = \hat{u}_i - \tilde{u}_i$. As already mentioned, the initial coarse grid mass fluxes, $\tilde{\dot{m}}$, are obtained by summing the corresponding fine grid mass fluxes. These change only if the velocities change; we assume that the fine grid mass fluxes are mass-conserving at the beginning of the multigrid cycle; if not, the mass imbalance can be included in the pressure-correction equation on the coarse grid.

It is important to take care that the implementation of boundary conditions also fulfills the consistency requirements. For example, if a symmetry boundary condition is implemented by setting the boundary value equal to the value at the near-boundary node, the restriction operator cannot calculate the boundary value of $\tilde{\phi}$ by interpolating the fine grid boundary values; it must calculate $\tilde{\phi}$ at all inner nodes and then apply the boundary condition to it, i.e. set $\tilde{\phi}$ at the symmetry boundary equal to $\tilde{\phi}$ at the near-boundary nodes. If the boundary condition is applied to $\hat{\phi}$, then ϕ' would not be the same at the boundary and next-to-boundary nodes, and a gradient of ϕ' would be passed to the finer grid. Then the solution on the fine grid cannot be converged beyond a certain limit. A similar situation can occur due to inconsistencies in treating other boundary conditions, but we shall not list all the possibilities here. It is important to assure that the iteration errors can be reduced to machine accuracy (even though this criterion will not be used when the code is put into production); if this is not possible, something is wrong!

Other strategies (e.g., W-cycles) may be used for cycling between the grids. Efficiency may be improved by basing the decision to switch from one grid to another on the rate of convergence. The simplest choice is the V-cycle described above with a fixed number of iterations on each grid level. The behavior of the FMG method for the V-cycle with typical numbers of iterations at each level is schematically shown in Fig. 11.6. The optimum choice of parameters is problem dependent, but their effect on performance is not as dramatic as for the single-grid method. Details of multigrid methods can be found in the book by Hackbusch (1985).
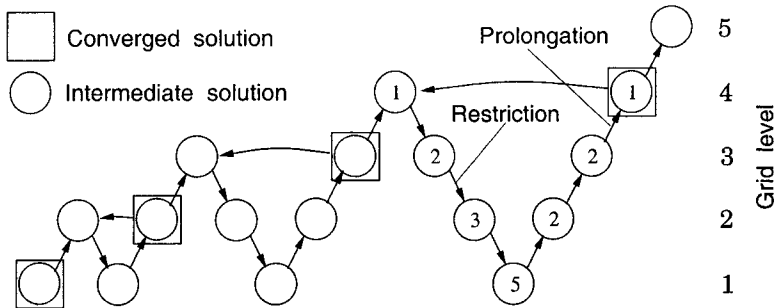
Fig. 11.6. Schematic presentation of FMG scheme using V-cycles, showing typical numbers of outer iterations at different stages in one cycle

The multigrid method can be applied to unstructured grids as well as structured grids. In FV methods, one usually joins fine grid CVs to produce coarse grid CVs; the number of fine CVs per coarse CV may differ, depending on the shape of CVs (tetrahedra, pyramids, prisms, hexahedra etc.). The multigrid idea can even be used if the coarse and fine grids are not related by

systematic refinement or coarsening – the grids may be arbitrary; it is only important that the solution domain and boundary conditions be the same on all levels, and that one grid is substantially coarser than the other (otherwise computational efficiency is not improved). The restriction and prolongation operators are then based on generic interpolation methods; such multigrid methods are called *algebraic multigrid methods* (see e.g. Raw, 1995, and Weiss et al., 1999).

For unsteady flows with implicit methods and small time steps, the outer iterations usually converge very rapidly (residual reduction by an order of magnitude per outer iteration), so multigrid acceleration is not necessary. The biggest savings are achieved for fully elliptic (diffusion-dominated) problems, and the smallest for convection-dominated problems (Euler equations). Typical acceleration factors range from 10 to 100 when five grid levels are used. An example is given below.

When computing turbulent flows with the k–ε turbulence model, interpolation may produce negative values of k and/or ε in the early cycles of a multigrid method; the corrections then have to be limited to maintain positivity. In problems with variable properties, the properties may vary by several orders of magnitude within the solution domain. This strong non-linear coupling of the equations may cause the multigrid method to become unstable. It may be better to update some quantities (e.g. the turbulent viscosity in the $k - \epsilon$ turbulence model) only on the finest grid and keep them constant within a multigrid cycle.
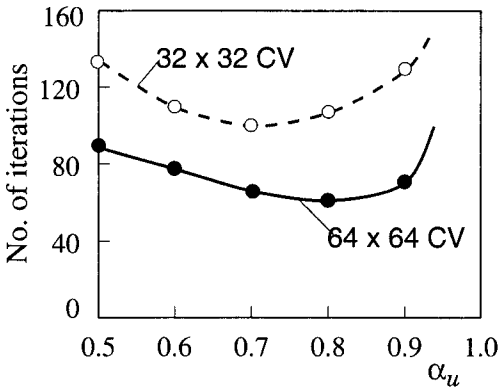
Fig. 11.7. Number of outer iterations on the finest grid in a multigrid method as a function of the under-relaxation factor α_u for the lid-driven cavity flow at Re = 1000

Under-relaxation factors are relatively unimportant in multigrid methods for laminar flows; the methods are less sensitive to these parameters than the single grid method. In Fig. 11.7 we show the dependence of the number of required outer iterations to solve the lid-driven cavity problem at Re = 1000 on the under-relaxation factor for velocity (using optimum under-relaxation of pressure correction, see Sect. 7.8) for two grids. The number of iterations varies by about 30% in the range of α_u between 0.5 and 0.9, while for the

single grid method, the variation was by a factor of five to seven (higher for finer grids; see Fig. 7.15). However, for turbulent flows, heat transfer, etc., under-relaxation may affect the multigrid method substantially.

The full multigrid method described above provides solutions on all grids. The cost of solving on all of the coarse grids is about 30% of the cost of the finest grid solution. If the solution process were started on the finest grid with zero fields, more total effort is needed than if the FMG approach is used. The savings resulting from more accurate initial fields usually outweigh the cost needed to obtain them. In addition, having solutions on a series of grids allows evaluation of discretization errors, as discussed in Sect. 3.9 and provides a basis for grid refinement. The grid refinement process may be stopped when the desired accuracy is achieved. Also, Richardson extrapolation may be used.

The multigrid approach to accelerating outer iterations described above can be applied to any solution method for the Navier-Stokes equations. Vanka (1986) applied it to the point-coupled solution method; Hutchinson and Raithby (1986) and Hutchinson et al. (1988) use it with a line-coupled solution technique. Methods of the SIMPLE type and fractional-step methods are also well suited for multigrid acceleration; see e.g. Hortmann et al. (1990) and Thompson and Ferziger (1989). The role of the *smoother* is now taken by the basic algorithm (e.g. SIMPLE); the linear equation solver plays a minor role.

In Table 11.1 we compare the numbers of outer iterations needed to solve the 2D lid-driven cavity flow problem at Reynolds numbers Re = 100 and Re = 1000 using different solution strategies. SG denotes the single-grid method with a zero initial field. PG denotes the prolongation scheme, in which the solution from the next coarser grid is used to provide the initial field. MG denotes the multigrid method using V-cycles, with the finest grid having zero initial fields. Finally, FMG denotes the multigrid method described above, which can be considered a combination of the PG and MG schemes.

The results show that, for the Re = 100 case, MG and FMG need about the same number of outer iterations on the finest grid level – a quality initial guess does not save much. For the single-grid scheme, the savings are substantial: on the 128×128 CV grid, the number of iterations is reduced by a factor of 3.5. The multigrid method reduces the number of iterations on that grid by a factor of 15; this factor increases as the grid is refined. The number of iterations remains constant in the MG and FMG methods from the third grid onwards, while it increases by a factor of four in SG and by a factor of 2.5 in the PG scheme.

For high Reynolds number flows, the situation changes a bit. SG needs fewer iterations than at Re = 100, except on coarse grids, on which the use of CDS slows convergence. PG reduces the number of iterations by less than a factor of two. MG needs about twice as many iterations as it did for

Table 11.1. Numbers of iterations and computing times required by various versions of the solution method to reduce the L_1 residual norm by four orders of magnitude when solving the lid-driven 2D cavity flow problem ($\alpha_u = 0.8$, $\alpha_p = 0.2$, non-uniform grid; for PG and FMG, CPU-times include computing times on all coarser grids)

Re	Grid	No. outer Iter.				CPU-Time			
		SG	PG	MG	FMG	SG	PG	MG	FMG
100	8^2	58	58	58	58	0.3	0.3	0.3	0.3
	16^2	61	51	47	45	0.9	1.2	1.4	1.5
	32^2	156	99	41	41	9.1	7.0	4.0	5.0
	64^2	555	256	40	40	140.8	71.1	13.0	16.9
	128^2	2119	620	40	40	2141.9	702.6	50.9	66.5
	256^2	-	-	40	40	-	-	242.2	293.8
1000	8^2	124	124	124	124	0.5	0.5	0.5	0.5
	16^2	156	162	123	132	2.2	2.5	2.8	2.9
	32^2	250	288	132	132	14.0	19.2	11.2	13.8
	64^2	433	400	93	73	97.0	120.7	32.0	38.5
	128^2	1352	725	83	41	1383.4	851.1	121.5	92.4
	256^2	-	-	83	31	-	-	512.9	278.8

Re = 100. However, FMG becomes more efficient as the grid is refined – the number of iterations required is actually lower on a 256 × 256 CV grid than for Re = 100. This is typical behavior of the multigrid method applied to the Navier-Stokes equations. The FMG approach is usually the most efficient one.

Results similar to those presented in Table 11.1 are also obtained for the 3D cavity flow; see Lilek et al. (1997a) for details.

In Fig. 11.8 the reduction of residual norm (the sum of absolute values of the residual over all CVs) of the turbulent kinetic energy k are shown for the computation of turbulent flow in a segment of a tube bundle made with the $k - \epsilon$ model. The curves are typical for the MG and SG methods. In practical applications, reduction of residuals by three to four orders of magnitude usually suffices. Here the residuals were reduced more than necessary, to show that the rate of convergence does not deteriorate. The saving in computing time varies from application to application: it is lower for convection-dominated than for diffusion-dominated flows.

11.4 Adaptive Grid Methods and Local Grid Refinement

Issues of accuracy have plagued computational fluid dynamics from its inception. There are many published results with significant errors. Tests intended to determine model validity have sometimes proven inconclusive because the
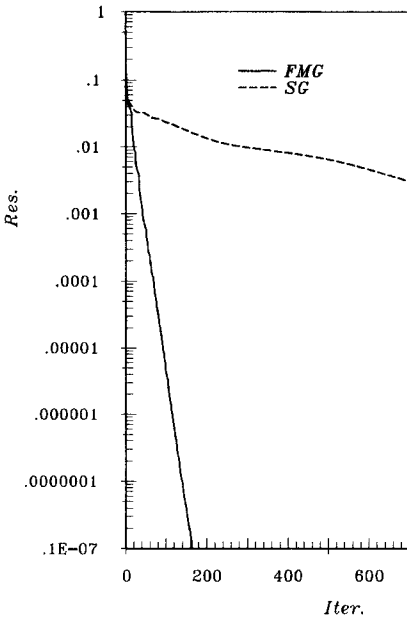
Fig. 11.8. Reduction of residual norm for the turbulent kinetic energy k in a calculation of flow in a tube bundle segment; the finest grid had 176×48 CV, five levels were used (from Lilek, 1995)

numerical errors were greater than the effects of the model. Recent comparative studies in which the same problem was solved by different groups using different codes reveal that the differences between solutions obtained using different codes with the same models are often larger than the differences between solutions obtained using the same code and different models (Bradshaw et al., 1994; Rodi et al., 1995). These differences can only be due to numerical error or user mistakes, if the models *are* really the same (it is not unusual that supposedly same models turn out to be different due to different interpretation, implementation, or boundary treatment). For valid comparisons of models, it is critical that errors be estimated and reduced.

The first essential is a method of estimating errors; the Richardson method given earlier is a good choice. A method of estimating the error without the need to perform calculations on two grids is to compare the fluxes through CV faces that result from the discretization method employed in the solution and a more accurate (higher-order) method. This is not as accurate as the Richardson method, but it does serve the purpose of indicating where the errors are large. Since the higher-order approximation is usually more complex, it is used only to calculate the fluxes after a converged solution with the base method has been obtained. For example, one can fit a polynomial of degree three through the cell centers on either side of a particular face and use the variable values and gradients at these two cell centers to find the coefficients of the polynomial (see Sect. 4.4.4 for an example). Assume that, if this approximation had been used, the exact solution Φ would have been obtained instead of the solution ϕ that resulted from the usual approximations. The

difference between fluxes computed using the cubic (F_k^Φ) and linear fits (F_k^ϕ) should be added to the discretized equation as an additional source term to recover the "exact" solution. If we define the discretization error ϵ^d and the source term τ (which is often called *tau-error*) as follows:

$$\epsilon^\mathrm{d} = \Phi - \phi \quad \text{and} \quad \tau_\mathrm{C} = \sum_k (F_k^\Phi - F_k^\phi) \, , \tag{11.11}$$

we obtain the following link between an estimate of the discretization error and the tau-error:

$$A_\mathrm{C}\epsilon_\mathrm{C}^\mathrm{d} + \sum_k A_k \epsilon_k^\mathrm{d} = \tau_\mathrm{C} \, . \tag{11.12}$$

Instead of solving this equation system for ϵ^d, it is often sufficient to simply normalize the τ_C by $A_\mathrm{C}\phi_\mathrm{C}$ and use this quantity as an estimate of the discretization error; this corresponds to performing one Jacobi iteration on the system of Eqs. (11.12) starting with zero initial values. The reason is that the above analysis is only approximate and the computed quantity is rather an *indication* than *estimation* of the discretization error. For more details and examples of application of this method of error estimation, see Muzaferija and Gosman (1997).

If the error estimate at a particular grid point is larger than a prescribed level, the cell is labeled for refinement. The boundaries of the refinement region should be extended by some margin, which should be a function of the local mesh size; the width of two to four cells is usually a sensible choice.

Block-structured grids require that refinement be performed block-wise; non-matching interface capability is required if not all blocks are refined. For unstructured grids, local refinement can be cell-wise, as illustrated in Fig. 11.9. Otherwise, cells to be refined may be clustered and new blocks of refined grid be defined, as will be described later.

The objective is to make the error everywhere smaller than some tolerance δ, either in terms of the absolute error, $||\epsilon||$, or the relative error, $||\epsilon/\phi||$. This can be accomplished by using methods of differing accuracy, an approach commonly used in ordinary differential equation solvers, but this is rarely done. One can also refine the grid everywhere but this is wasteful. A more flexible choice is to refine the grid *locally* where the errors are large. Experienced users of CFD codes may generate grids that are fine where necessary and coarse elsewhere, so that they yield a nearly uniform distribution of discretization error. However, this is difficult to do, especially if the geometry contains small but important protrusions e.g. mirrors on cars, appendages on ships and other vessels, small inlets and outlets on walls of large chambers etc. In such cases, local grid refinement is essential.

Some authors perform calculations on the refined portion of the grid only, using boundary conditions taken from the coarse grid solution at the refinement interface. This is called the *passive* method because the solution on the
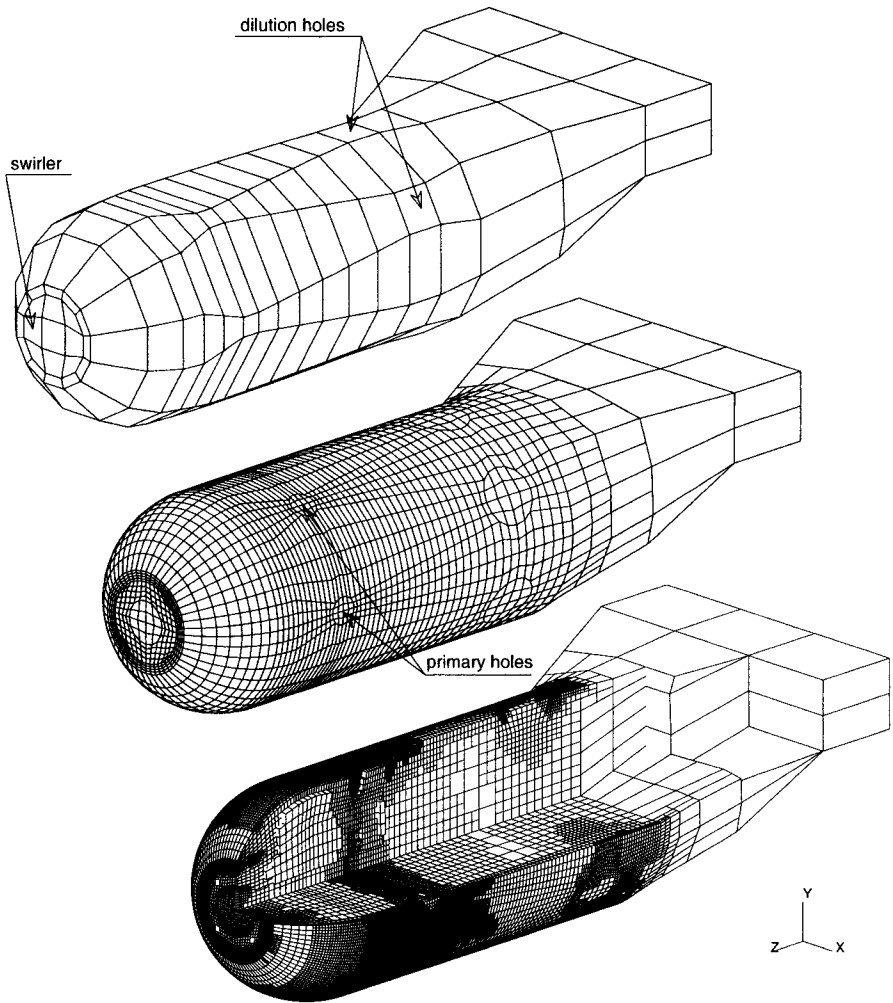
Fig. 11.9. Local grid refinement for the calculation of flow in a combustor: the coarsest grid (top), second refinement (middle) and final grid (bottom; fifth level); from Muzaferija and Gosman, 1997

unrefined part of the grid is not re-computed (see Berger and Oliger, 1984). This feature makes the method inappropriate for elliptic problems in which a change in conditions in any region may affect the solution everywhere. Methods that allow the influence of the refined grid solution to spread over the whole domain are called *active* methods. Such methods have been developed by Caruso et al. (1985) and Muzaferija (1994), among others.

One kind of active method (e.g. Caruso et al., 1985) proceeds exactly as the passive method with the important difference that the procedure is not complete when the fine grid solution has been computed. Rather, it is

necessary to compute a new coarse grid solution; this solution is not the one that would be computed on the coarse grid covering the entire domain but a smoothed version of the fine grid solution. To see what is needed, suppose that:

$$\mathcal{L}_h(\phi_h) = Q_h \tag{11.13}$$

is a discretization of the problem on a grid of size h; $\mathcal{L}$ represents the operator. To force the solution to be a smoothed version of the fine-grid solution in the region which has been refined we replace the coarse-grid problem by:

$$\mathcal{L}_{2h}(\phi_{2h}) = \begin{cases} \mathcal{L}_{2h}(\tilde{\phi}_h), & \text{in the refined region;} \\ Q_{2h}, & \text{in the remainder of the domain,} \end{cases} \tag{11.14}$$

where $\tilde{\phi}_h$ is the smoothed fine-grid solution (i.e. its representation on the coarse grid). The solution is then iterated between the coarse and fine grids until the iteration error is small enough; about four iterations usually suffice. Since the solution on each grid does not need to be iterated to final tolerance each time, this method costs only a little more than the passive method.

In another kind of active method (Muzaferija, 1994; Muzaferija and Gosman, 1997) the grids are combined into a single global grid, including the refined grid as well as the non-refined part of the original grid. This requires a solution method that allows CVs with arbitrary numbers of faces. CVs at an interface between refined and non-refined regions have more faces and neighbors than regular CVs, see Fig. 11.10. For the global conservation property of the FV method to be retained, the face of a non-refined CV on the refinement boundary has to be treated as two (in 2D, and four in 3D) separate sub-faces, each common to two CVs. In the discretization, the sub-faces are treated exactly like any other face between two CVs. The computer code needs to have a data structure that can handle this situation and the solver needs to be able to handle the irregular matrix structure that results. Conjugate gradient type solvers are a good choice; multigrid solvers with Gauss-Seidel smoothers can also be used with some limitations. The data structure can be optimized by storing cell-face and cell-volume related values in separate arrays. For a simple discretization scheme like the one described in Chap. 8, this is easily done: each cell face is common to two CVs so, for each face, one needs to store pointers to the nodes of neighbor CVs, surface vector components, and the matrix coefficients. The computer code for solving the flow problem is then the same for locally refined and standard grids; only the pre-processor needs adaptation to enable it to handle the data for locally refined grids.

If Chimera grids are used, the code needs no changes, but the interpolation coefficients and the nodes involved in interpolation need be redefined after each refinement.

As many levels of grid refinement as necessary can be used; usually at least three levels are required but as many as eight have been used. The advantage of the adaptive grid method is that, because the finest grid occupies only a
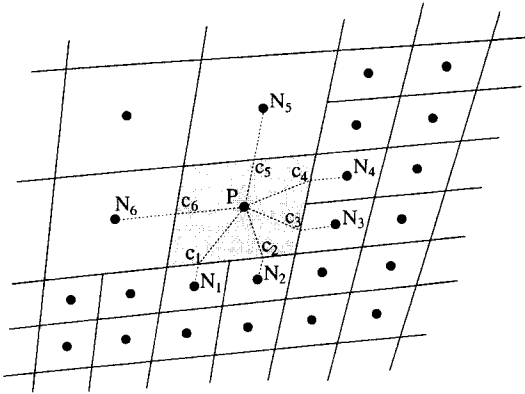
Fig. 11.10. A non-refined CV at the refinement interface: it has six faces (c_1, ..., c_6) in common with six neighbor CVs (N_1, ..., N_6)

small part of the domain, the total number of grid points is relatively small so both the cost of computation and the memory requirements are reduced enormously. Furthermore, it can be designed so that the user need not be an expert grid designer. Especially for flows around bluff bodies such as cars, airplanes, and ships, in which very fine grids are needed near the body and in the wake but coarse grids can be used elsewhere, local grid refinement is essential for accurate and efficient simulation.

Finally, these methods combine very well with the multigrid method. The nested grids can be regarded as the ones used in multigrid; the only important difference is that, because the coarse grids provide enough accuracy where refinement was not necessary, the finest grids do not cover the entire domain. In the non-refined region, equations to be solved remain the same for both levels. Since the largest cost of the multigrid method is due to iterations on the finest grid, the savings can be very large, especially in 3D. For further details, see Thompson and Ferziger (1989) and Muzaferija (1994).

In Fig. 11.9 an example of the use of locally refined grids to solve a complex flow problem is presented. Fluid is injected into a large combustion chamber through holes of different size. Uniform grid refinement would soon exhaust the computer memory. For example, refining the first grid which has 342 CV uniformly four times yields a grid with 1.4 million CVs; the locally refined grid shown in the figure has the same number of refinements and about 0.25 million CVs, about one sixth as many. The uniformly refined grid thus requires about six times as much memory, an even larger increase in computing time, and yields only slightly higher accuracy. In compressible flows, local grid refinement is necessary to resolve shocks efficiently.

11.5 Parallel Computing in CFD

The rapid growth in capability of single-processor computers has slowed in recent years. It now appears that further increases in speed will require mul-

tiple processors i.e., parallel computers. The advantage of parallel computers over classical vector supercomputers is scalability. They also use standard chips and are therefore cheaper to produce. Commercially available parallel computers may have thousands of processors, terabytes of memory and computing power measured in teraflops. However, algorithms designed for traditional serial machines may not run efficiently on parallel computers.

If parallelization is performed at the loop level (as is the case with auto-parallelizing compilers), Amdahl's law, which essentially says that the speed is determined by the least efficient part of the code, comes into play. To achieve high efficiency, the portion of the code that cannot be parallelized has to be very small.

A better approach is to subdivide the solution domain into sub-domains and assign each sub-domain to one processor. In this case the same code runs on all processors, on its own set of data. Since each processor needs data that resides in other sub-domains, exchange of data between processors and/or storage overlap is necessary.

Explicit schemes are relatively easy to parallelize, since all operations are performed on data from preceding time steps. It is only necessary to exchange the data at the interface regions between neighboring sub-domains after each step is completed. The sequence of operations and the results are identical on one and many processors. The most difficult part of the problem is usually the solution of the elliptic Poisson-like equation for the pressure.

Implicit methods are more difficult to parallelize. While calculation of the coefficient matrix and the source vector uses only 'old' data and can be efficiently performed in parallel, solution of the linear equation systems is not easy to parallelize. For example, Gauss elimination, in which each computation requires the result of the previous one, is very difficult to perform on parallel machines. Some other solvers can be parallelized and perform the same sequence of operations on n processors as on a single one, but they are either not efficient or the communication overhead is very large. We shall describe two examples.

11.5.1 Iterative Schemes for Linear Equations

The *red-black Gauss-Seidel* method is well suited for parallel processing. It was briefly described in Sect. 5.3.9 and consists of performing Jacobi iterations on two sets of points in an alternating manner. In 2D, the nodes are colored as on a checkerboard; thus, for a five point computational molecule, Jacobi iteration applied to a red point calculates the new value using data only from black neighbor nodes, and vice versa. The convergence properties of this solver are exactly those of the Gauss-Seidel method, which gave the method its name.

Computation of new values on either set of nodes can be performed in parallel; all that is needed is the result of the previous step. The result is exactly the same as on a single processor. Communication between processors

working on neighbor sub-domains takes place twice per iteration – after each set of data is updated. This local communication can be overlapped with computation of the new values. This solver is suitable only when used in conjunction with a multigrid method, as it is rather inefficient.
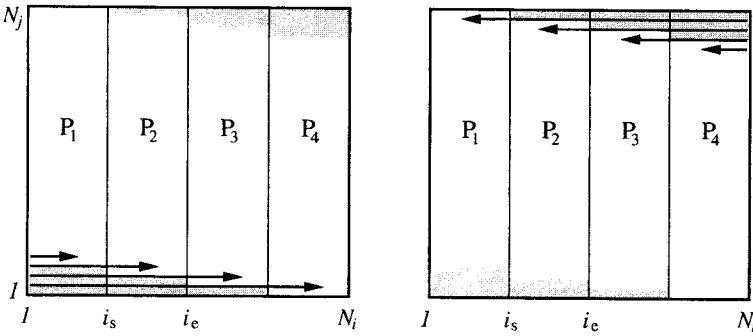
Fig. 11.11. Parallel processing in the SIP solver in the forward loops (left) and in the backward loop (right); shaded are regions of uneven load

ILU-type methods (e.g. the SIP-method presented in Sect. 5.3.4) are recursive, making parallelization less straightforward. In the SIP-algorithm, the elements of the L and U matrices, Eqs. (5.41), depend on the elements at the W and S nodes. One cannot start the calculation of the coefficients on a sub-domain, other than the one in the southwest corner, before data are obtained from its neighbors. In 2D, the best strategy is to subdivide the domain into vertical stripes i.e., use a 1D processor topology. Computation of L and U matrices and iteration can then be performed fairly efficiently in parallel (see Bastian and Horton, 1989). The processor for sub-domain 1 needs no data from other processors and can start immediately; it proceeds along its bottom or southernmost line. After it has calculated the elements for the rightmost node, it can pass those values to the processor for sub-domain 2. While the first processor starts calculation on its next line, the second one can compute on its bottom line. All n processors are busy when the first one has reaches the n-th line from bottom. When the first processor reaches the top boundary, it has to wait until the last processor, which is n lines behind, is finished; see Fig. 11.11. In the iteration scheme, two passes are needed. The first is done in the manner just described while the second is essentially its mirror image.

The algorithm is as follows:

for $j = 2$ to $N_j - 1$ do:
  receive $U_E(i_s - 1, j)$, $U_N(i_s - 1, j)$ from west neighbor;
  for $i = i_s$ to i_e do:
    calculate $U_E(i,j)$, $L_W(i,j)$, $U_N(i,j)$, $L_S(i,j)$, $L_P(i,j)$;
  end i;

```
      send U_E(i_e, j), U_N(i_e, j) to east neighbor;
   end j;

   for m = 1 to M do:
      for j = 2 to N_j - 1 do:
         receive R(i_s - 1, j) from west neighbor;
         for i = i_s to i_e do:
            calculate ρ(i, j), R(i, j);
         end i;
         send R(i_e, j) to east neighbor;
      end j;

      for j = N_j - 1 to 2 step -1 do:
         receive δ(i_e + 1, j) from east neighbor;
         for i = i_e to i_s step -1 do:
            calculate δ(i, j);
            update variable;
         end i;
         send δ(i_s, j) to west neighbor;
      end j;
   end m.
```

The problem is that this parallelization technique requires a lot of (fine grain) communication and there are idle times at the beginning and end of each iteration; these reduce the efficiency. Also, the approach is limited to structured grids. Bastian and Horton (1989) obtained good efficiency on transputer-based machines, which have a favorable ratio of communication to computation speed. With a less favorable ratio, the method would be less efficient.

The conjugate gradient method (without preconditioning) can be parallelized straightforwardly. The algorithm involves some global communication (gathering of partial scalar products and broadcasting of the final value), but the performance is nearly identical to that on a single processor. However, to be really efficient, the conjugate gradient method needs a good pre-conditioner. Since the best pre-conditioners are of the ILU-type (SIP is a very good pre-conditioner), the problems described above come into play again.

The above development shows that parallel computing environments require redesign of algorithms. Methods that are excellent on serial machines may be almost impossible to use on parallel machines. Also, new standards have to be used in assessing the effectiveness of a method. Good parallelization of implicit methods requires modification of the solution algorithm. The performance in terms of the number of numerical operations may be poorer than on a serial computer, but if the load carried by the processors is equalized and the communication overhead and computing time are properly matched, the modified method may be more efficient overall.

11.5.2 Domain Decomposition in Space

Parallelization of implicit methods is usually based on data parallelism or *domain decomposition*, which can be performed both in space and time. In spatial domain decomposition, the solution domain is divided into a certain number of sub-domains; this is similar to block-structuring of grids. In block-structuring the process is governed by the geometry of the solution domain, while, in domain decomposition, the objective is to maximize efficiency by giving each processor the same amount of work to do. Each sub-domain is assigned to one processor but more than one grid block may be handled by one processor; if so, we may consider all of them as one logical sub-domain.

As already noted, one has to modify the iteration procedure for parallel machines. The usual approach is to split the global coefficient matrix A into a system of diagonal blocks A_{ii}, which contain the elements connecting the nodes that belong to the ith sub-domain, and off-diagonal blocks or coupling matrices A_{ij} $(i \neq j)$, which represent the interaction of blocks i and j. For example, if a square 2D solution domain is split into four sub-domains and the CVs are numbered so that the members of each sub-domain have consecutive indices, the matrix has the structure shown in Fig. 11.12; a five-point molecule discretization is used in this illustration. The method described below is applicable to schemes using larger computational molecules; in this case, the coupling matrices are larger.
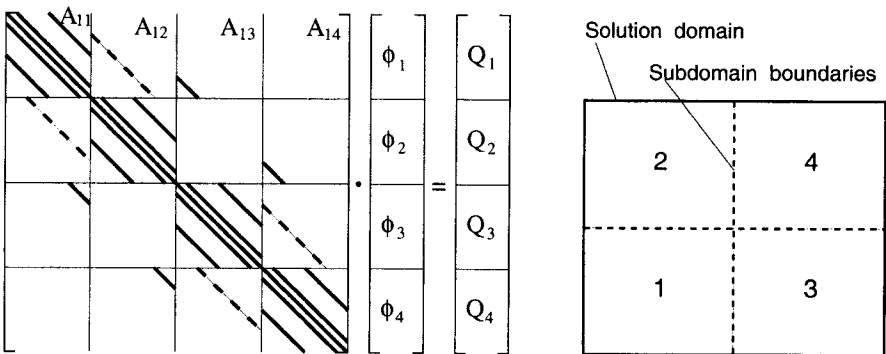
Fig. 11.12. Structure of the global coefficient matrix when a square 2D solution domain is subdivided into 4 sub-domains

For efficiency, the iterative solver for the inner iterations should have as little data dependency (data provided by the neighbors) as possible; data dependency may result in long communication and/or idle times. Therefore, the global iteration matrix is selected so that the blocks are de-coupled, i.e. $M_{ij} = 0$ for $i \neq j$. The iteration scheme on sub-domain i is then:

$$M_{ii}\phi_i^m = Q_i^{m-1} - (A_{ii} - M_{ii})\phi_i^{m-1} - \sum_j A_{ij}\phi_j^{m-1} \quad (j \neq i) \ . \quad (11.15)$$

The SIP solver is easily adapted to this method. Each diagonal block matrix M_{ii} is decomposed into L and U matrices in the normal way; the global iteration matrix $M = LU$ is not the one found in the single processor case. After one iteration is performed on each sub-domain, one has to exchange the updated values of the unknown ϕ^m so that the residual ρ^m can be calculated at nodes near sub-domain boundaries.

When SIP solver is parallelized in this way, the performance deteriorates as the number of processors becomes large; the number of iterations may double when the number of processors is increased from one to 100. However, if the inner iterations do not have to be converged very tightly, as is the case for some implicit schemes described in Chap. 7, parallel SIP can be quite efficient because SIP tends to reduce the error rapidly in the first few iterations. Especially if the multigrid method is used to speed-up the outer iterations, the total efficiency is quite high (80% to 90%; see Schreck and Perić, 1993, and Lilek et al., 1995, for examples). A 2D flow prediction code parallelized in this way is available via Internet; see appendix for details.
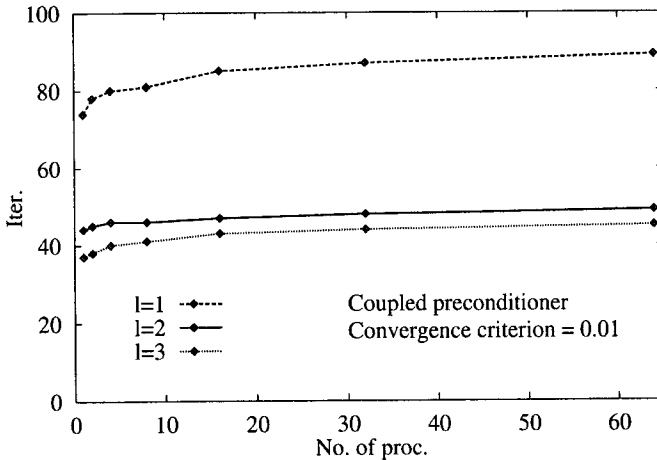
Fig. 11.13. Number of iterations in the ICCG solver as a function of the number of processors (uniform grid with 64^3 CVs, Poisson equation with Neumann boundary conditions, LC after each pre-conditioner sweep, l sweeps per CG iteration, residual norm reduced two orders of magnitude; from Seidl, 1997)

Conjugate gradient based methods can also be parallelized using the above approach. Below we present a pseudo-code for the preconditioned CG solver. It was found (Seidl et al., 1995) that the best performance is achieved by performing two pre-conditioner sweeps per CG iteration, on either single- or multi-processors. Results of solving a Poisson equation with Neumann bound-

ary conditions, which simulates the pressure-correction equation in CFD applications, are shown in Fig. 11.13. With one pre-conditioner sweep per CG iteration, the number of iterations required for convergence increases with the number of processors. However, with two or more pre-conditioner sweeps per CG iteration, the number of iterations remains nearly constant. However, in different applications one may obtain different behaviour.

- Initialize by setting: $k = 0$, $\phi^0 = \phi_{\text{in}}$, $\boldsymbol{\rho}^0 = \boldsymbol{Q} - A\phi_{\text{in}}$, $\boldsymbol{p}^0 = \boldsymbol{0}$, $s_0 = 10^{30}$
- Advance the counter: $k = k + 1$
- On each sub-domain, solve the system: $M\boldsymbol{z}^k = \boldsymbol{\rho}^{k-1}$
  LC: exchange $\boldsymbol{z}^k$ along interfaces
- Calculate: $s^k = \boldsymbol{\rho}^{k-1} \cdot \boldsymbol{z}^k$
  GC: gather and scatter s^k
  $\beta^k = s^k / s^{k-1}$
  $\boldsymbol{p}^k = \boldsymbol{z}^k + \beta^k \boldsymbol{p}^{k-1}$
  LC: exchange $\boldsymbol{p}^k$ along interfaces
  $\alpha^k = s_k / (\boldsymbol{p}^k \cdot A\boldsymbol{p}^k)$
  GC: gather and scatter α^k
  $\phi^k = \phi^{k-1} + \alpha^k \boldsymbol{p}^k$
  $\boldsymbol{\rho}^k = \boldsymbol{\rho}^{k-1} - \alpha^k A\boldsymbol{p}^k$
- Repeat until convergence.

To update the right hand side of Eq. (11.15), data from neighbor blocks is necessary. In the example of Fig. 11.12, processor 1 needs data from processors 2 and 3. On parallel computers with shared memory, this data is directly accessible by the processor. When computers with distributed memory are used, communication between processors is necessary. Each processor then needs to store data from one or more layers of cells on the other side of the interface. It is important to distinguish *local* (LC) and *global* (GC) communication.

Local communication takes place between processors operating on neighboring blocks. It can take place simultaneously between pairs of processors; an example is the communication within inner iterations in the problem considered above. GC means gathering of some information from all blocks in a 'master' processor and broadcasting some information back to the other processors. An example is the computation of the norm of the residual by gathering of the residuals from the processors and broadcasting the result of the convergence check. There are communication libraries, like PVM (Sunderam, 1990) or TCGMSG (Harrison, 1991), available on most parallel computers.

If the number of nodes allocated to each processor (i.e. the load per processor) remains the same as the grid is refined (which means that more processors are used), the ratio of local communication time to computing time will remain the same. We say that LC is fully scalable. However, the GC time increases when the number of processors increases, independent of the load per processor. The global communication time will eventually become larger

than the computing time as the number of processors is increased. Therefore, GC is the limiting factor in massive parallelism. Methods of measuring the efficiency are discussed below.

11.5.3 Domain Decomposition in Time

Implicit methods are usually used for solving steady flow problems. Although one is tempted to think that these methods are not well suited to parallel computing, they can be effectively parallelized by using domain decomposition in time as well as in space. This means that several processors simultaneously perform work on the same sub-domain for different time steps. This technique was first proposed by Hackbusch (1984).

Since none of the equations needs to be solved accurately within an outer iteration, one can also treat the 'old' variables in the discretized equation as unknowns. For a two-time-level scheme the equations for the solution at time step n can be written:

$$A^n \phi^n + B^n \phi^{n-1} = Q^n \, . \tag{11.16}$$

Since we are considering implicit schemes, the matrix and source vector may depend on the new solution, which is why they carry the index n. The simplest iterative scheme for solving simultaneously for several time steps is to decouple the equations for each time step and use old values of the variables where necessary. This allows one to start the calculation for the next time step as soon as the first estimate for the solution at the current time step is available, i.e. after one outer iteration is performed. The extra source term containing the information from the previous time step(s) is updated after each outer iteration, rather than being held constant as in serial processing. When the processor k, working on time level t_n, is performing its mth outer iteration, the processor $k-1$, working on time level t_{n-1}, is performing its $(m+1)$th outer iteration. The equation system to be solved by processor k in the mth outer iteration is then:

$$(A^n \phi^n)_k^m = (Q^n)_k^{m-1} - (B^n \phi^{n-1})_{k-1}^m \, . \tag{11.17}$$

The processors need to exchange data only once per outer iteration, i.e. the linear equation solver is not affected. Of course, much more data is transferred each time than in the method based on domain decomposition in space. If the number of time steps treated in parallel is not larger than the number of outer iterations per time step, using the lagged old values does not cause a significant increase in computational effort per time step. On the last time step in a parallel sequence, the term $(B^n \phi^{n-1})_{k-1}$ is included in the source term, as it does not change within iterations.

Figure 11.14 shows the structure of the matrix for a two-time-level scheme with simultaneous solution on four time steps. Time-parallel solution methods for CFD problems have been used by Burmeister and Horton (1991), Horton (1991), and Seidl et al. (1995), among others. The method can also be applied
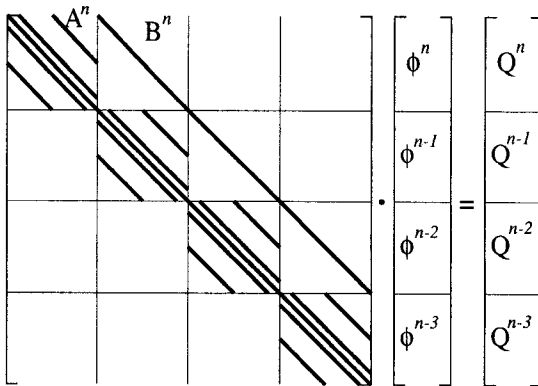
Fig. 11.14. Structure of the global coefficient matrix when four time time steps are calculated in parallel

to multilevel schemes; in that case the processors have to send and receive data from more than one time level.

11.5.4 Efficiency of Parallel Computing

The analysis of the performance of parallel programs is usually measured by the *speed-up* factor and *efficiency* defined by:

$$S_n = \frac{T_s}{T_n}\,, \quad E_n^{\text{tot}} = \frac{T_s}{n\,T_n}\,. \tag{11.18}$$

Here T_s is the execution time for the best serial algorithm on a single processor and T_n is the execution time for the parallelized algorithm using n processors. In general $T_s \neq T_1$, as the best serial algorithm may be different from the best parallel algorithm; one should not base the efficiency on the performance of the parallel algorithm executed on a single processor.

The speed-up is usually less than n (the ideal value), so the efficiency is usually less than 1 (or 100%). However, when solving coupled non-linear equations, it may turn out that solution on two or four processors is more efficient than on 1 processor so, in principle, efficiencies higher than 100% are possible (the increase is often due to the better use of cash memory when a smaller problem is solved by one processor).

Although not necessary, the processors are usually synchronized at the start of each iteration. Since the duration of one iteration is dictated by the processor with the largest number of CVs, other processors experience some idle time. Delays may also be due to different boundary conditions in different sub-domains, different numbers of neighbors, or more complicated communication.

The computing time T_s may be expressed as:

$$T_s = N^{\text{cv}} \tau i_s\,, \tag{11.19}$$

where N^{cv} is the total number of CVs, τ is the time per floating point operation and i_{s} is the number of floating point operations per CV required to reach convergence. For a parallel algorithm executed on n processors, the total execution time consists of computing and communication time:

$$T_n = T_n^{\text{calc}} + T_n^{\text{com}} = N_n^{\text{cv}} \tau i_n + T_n^{\text{com}} \,, \tag{11.20}$$

where N_n^{cv} is the number of CVs in the largest sub-domain and T_n^{com} is the total communication time during which calculation cannot take place. Inserting these expressions into the definition of the total efficiency yields:

$$E_n^{\text{tot}} = \frac{T_{\text{s}}}{n\,T_n} = \frac{N^{\text{cv}} \tau i_{\text{s}}}{n\,(N_n^{\text{cv}} \tau i_n + T_n^{\text{com}})} =$$

$$\frac{i_{\text{s}}}{i_n} \, \frac{1}{1 + T_n^{\text{com}}/T_n^{\text{calc}}} \, \frac{N^{\text{cv}}}{n\,N_n^{\text{cv}}} = E_n^{\text{num}} E_n^{\text{par}} E_n^{\text{lb}} \,. \tag{11.21}$$

This equation is not exact, since the number of floating point operations per CV is not constant (due to branching in the algorithm and the fact that boundary conditions affect only some CVs). However, it is adequate to identify the major factors affecting the total efficiency. The meanings of these factors are:

- E_n^{num} — The *numerical efficiency* accounts for the effect of the change in the number of operations per grid node required to reach convergence due to modification of the algorithm to allow parallelization;
- E_n^{par} — The *parallel efficiency* accounts for the time spent on communication during which computation cannot take place;
- E_n^{lb} — The *load balancing efficiency* accounts for the effect of some processors being idle due to uneven load.

When the parallelization is performed in both time and space, the overall efficiency is equal to the product of time and space efficiencies.

The total efficiency is easily determined by measuring the computing time necessary to obtain the converged solution. The parallel efficiency cannot be measured directly, since the number of inner iterations is not the same for all outer iterations (unless it is fixed by the user). However, if we execute a certain number of outer iterations with a fixed number of inner iterations per outer iteration on 1 and n processors, the numerical efficiency is unity and the total efficiency is then the product of the parallel and load balancing efficiencies. If the load balancing efficiency is reduced to unity by making all sub-domains equal, we obtain the parallel efficiency. Some computers have tools which allow operation counts to be performed; then the numerical efficiency can be directly measured.

For both space and time domain decomposition, all three efficiencies are usually reduced as the number of processors is increased for a given grid. This decrease is both non-linear and problem-dependent. The parallel efficiency is especially affected, since the time for LC is almost constant, the time for GC

increases, and the computing time per processor decreases due to reduced sub-domain size. For time parallelization, the time for GC increases while the LC and computing times remain the same when more time steps are calculated in parallel for the same problem size. However, the numerical efficiency will decrease disproportionately if the number of processors is increased beyond a certain limit (which depends on the number of outer iterations per time step). Optimization of the load balancing is difficult in general, especially if the grid is unstructured and local refinement is employed. There are algorithms for optimization, but they may take more time than the flow computation!

Parallel efficiency can be expressed as a function of three main parameters:

- set-up time for data transfer (called *latency time*);
- data transfer rate (usually expressed in Mbytes/s);
- computing time per floating point operation (usually expressed in Mflops).

For a given algorithm and communication pattern, one can create a model equation to express the parallel efficiency as a function of these parameters and the domain topology. Schreck and Perić (1993) presented such a model and showed that the parallel efficiency can be fairly well predicted. One can also model the numerical efficiency as a function of alternatives in the solution algorithm, the choice of solver and the coupling of the sub-domains. However, empirical input based on experience with similar flow problems is necessary, since the behavior of the algorithm is problem-dependent. These models are useful if the solution algorithm allows alternative communication patterns; one can choose the one most suitable for the computer used. For example, one can exchange data after each inner iteration, after every second inner iteration, or only after each outer iteration. One can employ one, two, or more pre-conditioner iterations per conjugate gradient iteration; the pre-conditioner iterations may include local communication after each step or only at the end. These options affect both the numerical and parallel efficiency; a trade-off is necessary to find an optimum.

Combined space and time parallelization is more efficient than pure spatial parallelization because, for a given problem size, the efficiency goes down as the number of processors increases. Table 11.5.4 shows results of the computation of the unsteady 3D flow in a cubic cavity with an oscillating lid at a maximum Reynolds number of 10^4, using a $32 \times 32 \times 32$ CV grid and a time step of $\Delta t = T/200$, where T is the period of the lid oscillation. When sixteen processors were used with four time steps calculated in parallel and the space domain decomposed into four sub-domains, the total numerical efficiency was 97%. If all processors are used solely for spatial or temporal decomposition, the numerical efficiency drops below 70%.

Communication between processors halts computation on many machines. However, if the communication and computation could take place simultaneously (which is possible on some new parallel computers), many parts of the solution algorithm could be rearranged to take advantage of this. For example, while LC takes place in the solver, one can do computation in the interior of

Table 11.2. Numerical efficiency for various domain decompositions in space and time for the calculation of cavity flow with an oscillating lid

Decomposition in space and time $x \times y \times z \times t$	Mean number of outer iterations per time step	Mean number of inner iterations per time step	Numerical efficiency (in %)
$1 \times 1 \times 1 \times 1$	11.3	359	100
$1 \times 2 \times 2 \times 1$	11.6	417	90
$1 \times 4 \times 4 \times 1$	11.3	635	68
$1 \times 1 \times 1 \times 2$	11.3	348	102
$1 \times 1 \times 1 \times 4$	11.5	333	104
$1 \times 1 \times 1 \times 8$	14.8	332	93
$1 \times 1 \times 1 \times 12$	21.2	341	76
$1 \times 2 \times 2 \times 4$	11.5	373	97

the sub-domain. With time parallelism, one can assemble the new coefficient and source matrices while LC is taking place. Even the GC in a conjugate gradient solver, which appears to hinder execution, can be overlapped with computation if the algorithm is rearranged as suggested by Demmel et al. (1993). Convergence checking can be skipped in the early stages of computation, or the convergence criterion can be rearranged to monitor the residual level at the previous iteration, and one can base the decision to stop on that value and the rate of reduction.

Perić and Schreck (1995) analyzed the possibilities of overlapping communication and computation in more detail and found that it can significantly improve parallel efficiency. New hard- and software are likely to allow concurrency of computation and communication, so one can expect that parallel efficiency can be optimized. One of the main concerns for the developers of parallel implicit CFD algorithms is numerical efficiency. It is essential that the parallel algorithm not need many more computing operations than the serial algorithm for the same accuracy. Results show that parallel computing can be efficiently used in CFD. The use of workstation clusters is especially useful with this respect, as they are available to almost all users and big problems are not solved all the time. It is expected that most computers (PCs, workstations and mainframes) will be multiprocessor machines in the future; it is therefore essential to have parallel processing in mind when developing new solution methods.

12. Special Topics

12.1 Introduction

Fluid flows may include a broad range of additional physical phenomena that take the subject far beyond the single-phase non-reacting flows that have been the focus of this work up to this point. Many types of physical processes may occur in flowing fluids. Each of these may interact with the flow to produce an amazing range of new phenomena. Almost all of these processes occur in important applications. Computational methods have been applied to them with varying degrees of success.

The simplest element that can be added to a flow is a scalar quantity such as the concentration of a soluble chemical species or temperature. The case in which the presence of the scalar quantity does not affect the properties of the fluid has already been treated in earlier chapters; in such a case, we speak of a passive scalar. In a more complex case, the density and viscosity of the fluid may be modified by the presence of the scalar and we have an active scalar. In a simple example, the fluid properties are functions of temperature or the concentration of the species. This field is known as heat and mass transfer.

In other cases, the presence of a dissolved scalar or the physical nature of the fluid itself cause the fluid to behave in way that the stress in not related to the strain rate by the simple Newtonian relationship (1.9). In some fluids, the viscosity becomes a function of the instantaneous strain rate and we speak of shear-thinning or shear-thickening fluids. In more complex fluids, the stress is determined by an additional set of non-linear partial differential equations. We then say that the fluid is viscoelastic. Many polymeric materials, including biological ones, exhibit this kind of behavior, giving rise to unexpected flow phenomena. This is the field of non-Newtonian fluid mechanics.

Flows may contain various kinds of interfaces. These may be due to the presence of a solid body in the fluid. In simple cases of this kind, it is possible to transform to a coordinate system moving with the body and the problem is reduced to one of the kind treated earlier, albeit in a complex geometry. In other problems, there may be bodies that move with respect to each other and there is no choice but to introduce a moving coordinate system. A particularly important and difficult case of this kind is one in which the surface is deformable. Surfaces of bodies of liquid are examples on this type.

In still other flows, multiple phases may coexist. All of the possible combinations are of importance. The solid-gas case includes such phenomena as dust in the atmosphere, fluidized beds, and gas flow through a porous medium. In the solid-liquid category are slurries (in which the liquid is the continuous phase), again, porous medium flows. Gas-liquid flows include sprays (in which the gas phase is continuous) and bubbly flows (in which the reverse is true). Finally, there may be three-phase flows. Each of these cases has many sub-categories.

Chemical reaction may take place in flows and again, there are many individual cases. When the reacting species are dilute, the reaction rates may be assumed constant (they may, however, depend on temperature) and the reacting species are essentially passive scalars with respect to their effect on the flow. Examples of this kind are pollutant species in the atmosphere or the ocean. Another kind of reaction involves major species and releases a large amount of energy. This is the case of combustion. Still another example is that of airflow at high speeds; compressibility effects may lead to large temperature increases and the possibility of dissociation or ionization of the gas.

Geophysics and astrophysics also require the solution of the equations of fluid motion. Other than plasma effects (discussed below), the new elements in these flows are the enormous scales compared to engineering flows. In meteorology and oceanography, rotation and stratification have a great influence on the behavior.

Finally, we mention that in plasmas (ionized fluids), electromagnetic effects play an important role. In this field, the equations of fluid motion have to be solved along with the equations of electro-magnetism (the Maxwell equations) and the number of phenomena and special cases is enormous.

In the remainder of this chapter, we shall describe methods for dealing with some, but not all, of these difficulties. We should point out that each of the topics mentioned above is an important sub-specialty of fluid mechanics and has a large literature devoted to it; references to textbooks in each area are given below. It is impossible to do justice to each of these topics in the space available here.

12.2 Heat and Mass Transfer

Of the three mechanisms of heat transfer – conduction, radiation, and convection – usually presented in courses on the subject, the last is most closely connected with fluid mechanics. The link is so strong that convective heat transfer may be regarded as a sub-area of fluid mechanics.

Steady heat conduction is described by Laplace's equation (or equations very similar to it) while unsteady conduction is governed by the heat equation; these equations are readily solved by methods presented in Chaps. 3, 4 and 6. A complication arises when the properties are temperature-dependent.

In such a case, the properties are calculated using the temperature on the current iteration, the temperature is updated, and the process is repeated. Convergence is usually nearly as rapid as in the fixed-property case.

In some applications, heat conduction in a solid needs to be considered along with convection in an adjacent fluid. Problems of this kind are called *conjugate heat transfer* problems and need to be solved by iterating between the equations describing the two types of heat transfer. Fully coupled methods have also been suggested.

Radiation involving solid surfaces has little connection with fluid mechanics (except in problems with multiple active mechanisms of heat transfer). There are interesting problems (for example, flows in rocket nozzles and combustors) in which both fluid mechanics and radiative heat transfer in the gas are important. The combination also occurs in astrophysical applications and in meteorology. We shall not deal with this type of problem here.

In laminar convective heat transfer, the dominant processes are *advection* (which we previously called convection!) in the streamwise direction and conduction in the direction normal to the flow. When the flow is turbulent, much of the role played by conduction in laminar flows is taken over by the turbulence and is represented by a turbulence model; these models are discussed in Chapter 9. In either case, interest generally centers on exchange of thermal energy with solid surfaces.

If the temperature differences are small (less than 5 K in water or 10 K in air), the variations of the fluid properties are not important and the temperature behaves as a passive scalar. Problems of this sort can be treated by methods described earlier in this book. Since the temperature is a passive scalar in this case, it can be computed after the computation of the velocity field has been completely converged, making the task much simpler. In the case in which the flow is driven by density differences, the latter must be taken into consideration. This can be done with the aid of the Boussinesq approximation described below.

Another important special case is that of heat transfer occurring in flows past bodies of smooth shape. In flows of this type, one can first compute the potential flow around the body and then use the pressure distribution obtained as input to a boundary layer code for the prediction of the heat transfer. If the boundary layer does not separate from the body, it is possible to compute these flows using the boundary-layer simplification of the Navier-Stokes equations (see, for example, Kays and Crawford, 1978, or Cebeci and Bradshaw, 1984). The boundary-layer equations are parabolic and can be solved in a matter of seconds (for the 2D case) or a minute or so (for the 3D case) on a modern workstation or personal computer. Methods for computing these flows have not been covered in detail in this work (but the general principles are found in Chaps. 3 to 7); the interested reader can find them in the works by Cebeci and Bradshaw (1984) and Patankar and Spalding (1977).

In the general case, temperature variations are significant. They affect the flow in two ways. The first is through the variation of the transport properties with temperature. These can be very large and must be taken into account but are not difficult to handle numerically. The important issue is that the energy and momentum equations are now coupled and must be solved simultaneously. Fortunately, the coupling is not usually so strong as to prevent solution of the equations in sequential fashion. On each outer iteration, the momentum equations are first solved using transport properties computed from the 'old' temperature field. The temperature field is updated after the solution of the momentum equations has been obtained for the new outer iteration and the properties are updated. This technique is very similar to the one for solving the momentum equations with a turbulence model described in Chapter 9.

Another effect of temperature variation is that density variation interacting with gravity, produces a body force that may modify the flow considerably and may be the principal driving force in the flow. In the latter case, we talk of *buoyancy-driven* or *natural convection* flow. The relative importance of *forced* convection and buoyancy effects is measured by the ratio of the Rayleigh and Reynolds numbers. The former is defined by:

$$Ra = \frac{g\,\Delta\rho\,L^3}{\rho_0 \nu \kappa} \,, \tag{12.1}$$

where g is the acceleration of gravity, $\Delta\rho$ is the density variation within the domain, ρ_0 is a reference density, and κ is the thermal diffusivity. If Re/Ra $> 10^4$, the effects of natural convection may be ignored.

In purely buoyancy-driven flows, if the density variations are small enough, it may be possible to ignore the density variations in all terms other than the body force in the vertical momentum equation. This is called the Boussinesq approximation and it allows the equations to be solved by methods that are essentially identical to those used for incompressible flow. An example was presented in Sect. 7.8.

Computation of flows in which buoyancy is important is usually made by methods of the type described above i.e. iteration of the velocity field precedes iteration for the temperature and density fields. Because the coupling between the fields may be quite strong, this procedure may converge more slowly than in isothermal flows. Solution of the equations as a coupled system increases the convergence rate at the cost of increased complexity of programming and storage requirements; see Galpin and Raithby (1986) for an example. The strength of the coupling also depends on the Prandtl number. It is stronger for fluids with high Prandtl numbers; for these fluids, the coupled solution approach yields much faster convergence than the sequential approach.

12.3 Flows With Variable Fluid Properties

Although we have dealt mainly with incompressible flows, the density, viscosity, and other fluid properties have been kept inside the differential operators. This allows the discretization and solution methods presented in the preceding chapters to be used to solve problems with variable fluid properties.

The variation in fluid properties is usually caused by temperature variation; pressure variation also affects the change of density. This kind of variation was considered in Chap. 10, where we dealt with compressible flows. However, there are many cases in which the pressure does not change substantially, but the temperature and/or concentration of solutes can cause large variation in fluid properties. Examples are gas flows at reduced pressure, flows in liquid metals (crystal growth, solidification and melting problems, etc.), and environmental flows of fluids stratified by dissolved salt.

Variations in density, viscosity, Prandtl number, and specific heat increase the non-linearity of the equations. The sequential solution method can be applied to these flows in the much the same way they are applied to flows with variable temperature. One recalculates the fluid properties after each outer iteration and treats them as known during the next outer iteration. If the property variation is significant the convergence may be slowed considerably. For steady flows, the multigrid method can result in a substantial speed-up; see Durst et al. (1992) for an example of application to metalorganic chemical vapor deposition problems, and Kadinski and Perić (1995) for application to problems involving thermal radiation.

Flows in the atmosphere and the oceans are special examples of variable density flows; they are discussed later.

12.4 Moving Grids

In many application areas the solution domain changes with time due to the movement of boundaries. The movement is determined either by external effects (as in piston-driven flows) or by calculation as part of the solution (for example, in free-surface flows). In either case, the grid has to move to accommodate the changing boundary. If the coordinate system remains fixed and the Cartesian velocity components are used, the only change in the conservation equations is the appearance of the relative velocity in convective terms; see Sect. 1.2. We describe here briefly how the equations for a moving grid system can be derived.

First consider the one dimensional continuity equation:

$$\frac{\partial \rho}{\partial t} + \frac{\partial (\rho v)}{\partial x} = 0 \, . \tag{12.2}$$

By integrating this equation over a control volume whose boundaries move with time, i.e. from $x_1(t)$ to $x_2(t)$, we get:

$$\int_{x_1(t)}^{x_2(t)} \frac{\partial \rho}{\partial t}\,\mathrm{d}x + \int_{x_1(t)}^{x_2(t)} \frac{\partial(\rho v)}{\partial x}\,\mathrm{d}x = 0\,. \tag{12.3}$$

The second term causes no problems. The first requires the use of Leibniz's rule and, as a result, Eq. (12.3) becomes:

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_{x_1(t)}^{x_2(t)} \rho\,\mathrm{d}x - \left[\rho_2\frac{\mathrm{d}x_2}{\mathrm{d}t} - \rho_1\frac{\mathrm{d}x_1}{\mathrm{d}t}\right] + \rho_2 v_2 - \rho_1 v_1 = 0\,. \tag{12.4}$$

The derivative $\mathrm{d}x/\mathrm{d}t$ represents the velocity with which the grid (integration boundary) moves; we denote it by v_b. The terms in square brackets have therefore a form similar to the last two terms involving fluid velocity, so we can rewrite the Eq. (12.3) as:

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_{x_1(t)}^{x_2(t)} \rho\,\mathrm{d}x + \int_{x_1(t)}^{x_2(t)} \frac{\partial}{\partial x}[\rho(v - v_\mathrm{b})]\,\mathrm{d}x = 0\,. \tag{12.5}$$

When the boundary moves with fluid velocity, i.e. $v_\mathrm{b} = v$, the second integral becomes zero and we have the Lagrangian mass conservation equation, $\mathrm{d}m/\mathrm{d}t = 0$.

The three-dimensional version of Eq. (12.4) (obtained using the 3D version of Leibniz's rule) gives:

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_\Omega \rho\,\mathrm{d}\Omega - \int_S \rho\frac{\mathrm{d}\boldsymbol{r}}{\mathrm{d}t}\cdot\boldsymbol{n}\,\mathrm{d}S + \int_S \rho\boldsymbol{v}\cdot\boldsymbol{n}\,\mathrm{d}S = 0\,, \tag{12.6}$$

or, in the notation used above:

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_\Omega \rho\,\mathrm{d}\Omega + \int_S \rho(\boldsymbol{v} - \boldsymbol{v}_\mathrm{b})\cdot\boldsymbol{n}\,\mathrm{d}S = 0\,. \tag{12.7}$$

In Sect. 1.2 we noted that the conservation laws can be transformed from the control mass to control volume form by using Eq. (1.4); this also leads to the above mass conservation equation. The same approach may be applied to any transport equation.

The integral form of the conservation equation for the ith momentum component takes the following form when the CV-surface moves with velocity $\boldsymbol{v}_\mathrm{b}$:

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_\Omega \rho u_i\,\mathrm{d}\Omega + \int_S \rho u_i(\boldsymbol{v} - \boldsymbol{v}_\mathrm{b})\cdot\boldsymbol{n}\,\mathrm{d}S = \int_S (\tau_{ij}\boldsymbol{i}_j - p\boldsymbol{i}_i)\cdot\boldsymbol{n}\,\mathrm{d}S + \int_\Omega b_i\,\mathrm{d}\Omega\,. \tag{12.8}$$

Conservation equations for scalar quantities are easily derived from the corresponding equations for a fixed CV by replacing the velocity vector in the convective term with the relative velocity $\mathbf{v} - \mathbf{v}_\mathrm{b}$.

Obviously, if the boundary moves with the same velocity as the fluid, the mass flux through the CV face will be zero. If this is true for all CV faces, then the same fluid remains within the CV and it becomes a *control mass;* we then have the Lagrangian description of fluid motion. On the other hand, if the CV does not move, the equations are those dealt with earlier.

When the location of the grid is known as a function of time, solution of the Navier-Stokes equations poses no new problems: we simply calculate the convective fluxes (e.g. the mass fluxes) using the relative velocity components at the cell faces. However, when the cell faces move, conservation of mass (and all other conserved quantities) is not necessarily ensured if the grid velocities are used to calculate the mass fluxes. For example, consider the continuity equation with implicit Euler time integration; for the sake of simplicity we assume that the CV is rectangular and that the fluid is incompressible and moves at constant velocity. Figure 12.1 shows the relative sizes of the CV at the old and new time levels. We also assume that the grid lines (CV faces) move with constant, but different velocities, so that the size of the CV grows with time.
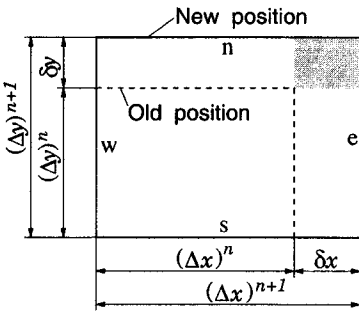
Fig. 12.1. A rectangular control volume whose size increases with time due to a difference in the grid velocities at its boundaries.

The discretized continuity equation for the CV shown in Fig. 12.1 with the implicit Euler scheme reads:

$$\frac{\rho\,[(\Delta\Omega)^{n+1} - (\Delta\Omega)^n]}{\Delta t} + \rho\,[(u - u_\mathrm{b})_\mathrm{e} - (u - u_\mathrm{b})_\mathrm{w}]^{n+1}(\Delta y)^{n+1} +$$
$$\rho\,[(v - v_\mathrm{b})_\mathrm{n} - (v - v_\mathrm{b})_\mathrm{s}]^{n+1}(\Delta x)^{n+1} = 0\;, \quad (12.9)$$

where u and v are the Cartesian velocity components. Since we assume that the fluid moves with a constant velocity, the contribution of fluid velocity in the above equation cancels out – only the difference in grid velocities remains:

$$\frac{\rho}{\Delta t}[(\Delta\Omega)^{n+1} - (\Delta\Omega)^n] - \rho(u_\mathrm{b,e} - u_\mathrm{b,w})(\Delta y)^{n+1} -$$

$$\rho(v_{\mathrm{b,n}} - v_{\mathrm{b,s}})(\Delta x)^{n+1} = 0 \, . \tag{12.10}$$

Under the assumptions made above, the difference in grid velocities at the opposite CV sides can be expressed as (see Fig. 12.1):

$$u_{\mathrm{b,e}} - u_{\mathrm{b,w}} = \frac{\delta x}{\Delta t} \, , \quad v_{\mathrm{b,n}} - v_{\mathrm{b,s}} = \frac{\delta y}{\Delta t} \, . \tag{12.11}$$

By substituting these expressions into Eq. (12.10) and noting that $(\Delta\Omega)^{n+1} = (\Delta x\, \Delta y)^{n+1}$ and $(\Delta\Omega)^n = [(\Delta x)^{n+1} - \delta x][(\Delta y)^{n+1} - \delta y]$, we find that the discretized mass conservation equation is not satisfied – there is a mass source

$$\delta\dot{m} = \frac{\rho\, \delta x\, \delta y}{\Delta t} = \rho(u_{\mathrm{b,e}} - u_{\mathrm{b,w}})(v_{\mathrm{b,n}} - v_{\mathrm{b,s}})\, \Delta t \, . \tag{12.12}$$

The same error (with opposite sign) is obtained with the explicit Euler scheme. For constant grid velocities it is proportional to the time step size, i.e. it is a first-order discretization error. One might think that this is not a problem, since the scheme is only first-order accurate in time; however, artificial mass sources may accumulate with time and cause serious problems. The error disappears if only one set of grid lines moves, or if the grid velocities are equal at opposite CV sides.

Under the above assumptions, both the Crank-Nicolson and three-time-level implicit scheme satisfy the continuity equation exactly. More generally, when the fluid and/or grid velocities are not constant, these schemes can also produce artificial mass sources.

Mass conservation can be obtained by enforcing the so-called *space conservation law* (SCL) which can be thought of as the continuity equation in the limit of zero fluid velocity:

$$\frac{\mathrm{d}}{\mathrm{d}t} \int_{\Omega} \mathrm{d}\Omega - \int_{S} \boldsymbol{v}_{\mathrm{b}} \cdot \boldsymbol{n}\, \mathrm{d}S = 0 \, . \tag{12.13}$$

This equation describes the conservation of space when the CV changes its shape and/or position with time.

In what follows the implicit Euler scheme for time integration is used for illustration; the extension to higher-order schemes is straightforward, but more complicated. For spatial integration we use the midpoint rule and central-difference schemes.

Equation (12.13) reads, in discretized form:

$$\frac{(\Delta\Omega)^{n+1} - (\Delta\Omega)^n}{\Delta t} = \sum_c (\boldsymbol{v}_{\mathrm{b}} \cdot \boldsymbol{n})_c\, S_c \, , \quad c = \mathrm{e, w, n, s}, \ldots \tag{12.14}$$

The difference between the 'new' and the 'old' CV volume can be expressed as the sum of volumes $\delta\Omega_c$ swept by the CV faces during the time step, see Fig. 12.2:
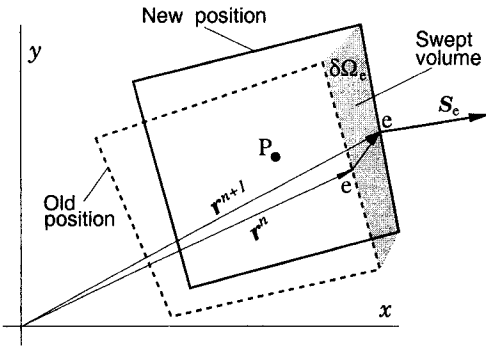
Fig. 12.2. A typical 2D CV at two time steps and the volume swept by a cell face.

$$\frac{(\Delta\Omega)^{n+1} - (\Delta\Omega)^n}{\Delta t} = \frac{\sum_c \delta\Omega_c}{\Delta t} . \tag{12.15}$$

By comparing these two equations, we see that the volume swept by one cell face is:

$$\dot{\Omega}_c = (\boldsymbol{v}_\mathrm{b} \cdot \boldsymbol{n})_c \, S_c = \frac{\delta\Omega_c}{\Delta t} . \tag{12.16}$$

The grid movement affects only the mass fluxes. When the CV position at all times is known, the grid velocity $\boldsymbol{v}_\mathrm{b}$ can be calculated explicitly. At the cell face center:

$$\boldsymbol{v}_{\mathrm{b},c} \approx \frac{\boldsymbol{r}_c^{\,n+1} - \boldsymbol{r}_c^{\,n}}{\Delta t} . \tag{12.17}$$

When the grid moves in only one direction, this approach causes no problems. It is also possible to transform equations to a moving coordinate system (Gosman, 1984). However, if the grid moves in more than one direction, it is difficult to ensure mass conservation using expressions like (12.17); artificial mass sources may be generated, as demonstrated by Demirdžić and Perić (1988). By computing the volumes defined by the cell face positions at each time step, these errors can be avoided, even if the time step is very large. The mass flux through a cell face 'c' can therefore be calculated as:

$$\dot{m}_c = \int_{S_c} \rho(\boldsymbol{v} - \boldsymbol{v}_\mathrm{b}) \cdot \boldsymbol{n} \, \mathrm{d}S \approx \rho_c(\boldsymbol{v} \cdot \boldsymbol{n})_c S_c - \rho_c \dot{\Omega}_c . \tag{12.18}$$

In a sequential solution method, the mass fluxes are treated as known in all other conservation equations, so these equations may be treated as they were on a stationary grid. Only the continuity equation requires special attention. For the implicit Euler scheme, the discretized continuity equation reads:

$$\frac{(\rho\,\Delta\Omega)^{n+1} - (\rho\,\Delta\Omega)^n}{\Delta t} + \sum_c \dot{m}_c = 0 , \quad c = \mathrm{e, w, n, s}, \ldots \tag{12.19}$$

The unsteady term has to be treated in a way that is consistent with the space conservation law. For incompressible flows, the contribution of the grid movement to the mass fluxes has to cancel the unsteady term, i.e. the mass conservation equation reduces to:

$$\int_S \rho \boldsymbol{v} \cdot \boldsymbol{n} \, \mathrm{d}S = 0 \, . \tag{12.20}$$

The discretization method must ensure that the unsteady term and the mass fluxes satisfy this equation if strict mass conservation is to be obtained. If the volume change and mass fluxes are calculated as above, this conservation is assured. Therefore, for incompressible flows, grid movement does not affect the pressure-correction equation.
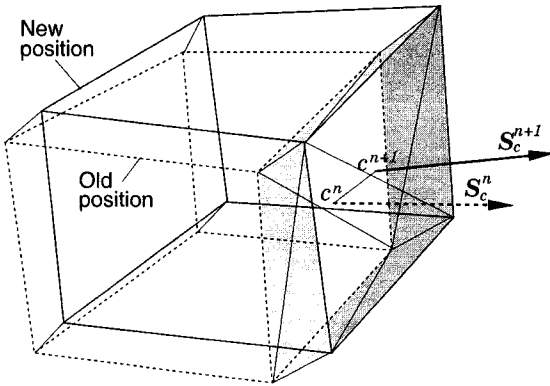
Fig. 12.3. On the calculation of a volume swept by a cell face of a 3D CV; shaded are surfaces common to neighbor CVs.

In three dimensions, one has to be careful in calculating the volumes swept by cell faces. Because the cell face edges may turn, the calculation of the swept volume requires triangulation of the shaded surfaces in Fig. 12.3. The volume can then be calculated using approach described in Sect. 8.6.4. However, as the shaded surfaces are common to two CVs, one has to ensure that they are triangulated in the same way for both CVs to assure space conservation.

Extension to higher-order schemes is straightforward. For example, discretization of the SCL, Eq. (12.13) by the Crank-Nicolson scheme leads to (see Eq. (12.16)):

$$\dot{\Omega}_c = \frac{1}{2} \left[(\boldsymbol{v}_\mathrm{b} \cdot \boldsymbol{n})_c^n \, S_c^n + (\boldsymbol{v}_\mathrm{b} \cdot \boldsymbol{n})_c^{n+1} \, S_c^{n+1} \right] = \frac{\delta \Omega_c}{\Delta t} \, . \tag{12.21}$$

The swept volume $\delta \Omega_c$ is calculated in the same way as for the implicit Euler scheme, but the mass flux is now calculated at the half step location:

$$\dot{m}_c = \frac{1}{2} \left[\rho_c^n (\boldsymbol{v} \cdot \boldsymbol{n})_c^n S_c^n + \rho_c^{n+1} (\boldsymbol{v} \cdot \boldsymbol{n})_c^{n+1} S_c^{n+1} \right] - \rho_c^* \dot{\Omega}_c \, . \tag{12.22}$$

For incompressible flows, the density is constant so $\rho_c^* = \rho =$ const. Compressible flows require special care in order to determine ρ_c^* such that both space and mass conservation equations are satisfied. The problem is that the mass conservation equation (12.19) contains the cell-center density in the unsteady term and cell-face densities in the mass fluxes, so that the evaluation of ρ_c^* is not trivial if the density varies rapidly in both space and time. If it does, small time steps are necessary so that the approximation $\rho_c^* = 1/2(\rho_c^n + \rho_c^{n+1})$ is sufficiently accurate.

The importance of taking the space (or geometric) conservation law into account in unsteady flows with moving boundaries has been recognized by many authors; see Thomas and Lombard (1979) and Demirdžić and Perić (1988) for a more detailed discussion. Using coordinate transformations or calculating the grid velocities from the motion of the cell-face center (see Eq. (12.17)) leads to artificial mass sources or sinks. The error depends on the time step and grid velocity, (see Eq. (12.12)). When the time step is small (which is usually the case with explicit schemes), the error is also small and is often neglected. However, care needs to be taken to avoid accumulation of the mass-imbalance error.

When the grid location is known at each time level, inclusion of grid movement in the solution procedure is simple; see Demirdžić and Perić (1990) for more details and an example. When the boundary movement is not known in advance, an iterative procedure has to be used at each time step (or outer iteration).

In some implicit time-integration methods (so-called *fully-implicit methods*), in which fluxes and source terms are computed only at the newest time level, grid motion can be ignored everywhere except near boundaries. Examples of such methods are the implicit Euler scheme and the three-time-level scheme, see Chap. 6. Since the fluxes are computed at time level t_{n+1}, we do not need to know where the grid was (or what shape the CVs had) at the previous time level t_n: instead of Eq. (12.8) we can use the usual equation for a space-fixed CV:

$$\frac{\partial}{\partial t}\int_{\Omega} \rho u_i \,\mathrm{d}\Omega + \int_S \rho u_i \boldsymbol{v}\cdot\boldsymbol{n}\,\mathrm{d}S = \int_S (\tau_{ij}\boldsymbol{i}_j - p\,\boldsymbol{i}_i)\cdot\boldsymbol{n}\,\mathrm{d}S + \int_{\Omega} b_i \,\mathrm{d}\Omega \,. \tag{12.23}$$

The two equations differ in the definition of the rate-of-change and convective terms: for a space-fixed CV convective fluxes are computed using fluid velocity, and time derivative represents the local rate of change at a fixed point in space (e.g. CV-center). On the other hand, for a moving CV convective fluxes are computed using relative velocity between fluid and CV-surface, and the time derivative expresses the rate of change in a volume whose location changes. If the CV-surface moves with the fluid velocity the same fluid re-

mains in the CV, it becomes the *material derivative* and the CV becomes a *control mass*.

Since solutions from previous time steps are not needed to compute surface and volume integrals, the grid can not only move but may also change its topology, i.e. both the number of CVs and their shape can change from one time step to another. The only term in which the old solution appears is the unsteady term, which requires that volume integrals over the *new* CV of some *old* quantities have to be approximated. If midpoint rule is used for this purpose, all we need to do is to interpolate the old solutions to the locations of the new CV-centers. One possibility is to compute gradient vectors at the center of each old CV and then, for each new CV-center, find the nearest center of an old CV and use linear interpolation to obtain the old value at the new CV-center:

$$\phi^{\text{old}}_{\text{C}^{\text{new}}} = \phi^{\text{old}}_{\text{C}^{\text{old}}} + (\text{grad}\,\phi)^{\text{old}}_{\text{C}^{\text{old}}} \cdot (\boldsymbol{r}_{\text{C}^{\text{new}}} - \boldsymbol{r}_{\text{C}^{\text{old}}}) \; . \tag{12.24}$$

Schneider (2000) investigated the effects of the order of interpolation on the overall accuracy of this approach and found – for a particular flow problem involving a moving indentation in a channel with laminar flow – that quadratic and cubic interpolation lead to better results but linear interpolation was acceptable on a sufficiently fine grid.

Near moving boundaries we have to account for the fact that the boundary moved during the time step and either displaced fluid or made space to be filled by fluid. For small motions this can be taken into account by prescribing inlet or outlet mass fluxes (or mass sources or sinks in the near-boundary CVs). A problem can arise if the CV moves more than its width in the direction of motion in one time step, since the center of a new CV may lie outside the old mesh. Thus, for grids which are fine near moving walls it may be desirable to use a moving grid and equations based on moving control volumes in the near-wall region, while away from walls the grid motion may be ignored, allowing for the grid to be re-generated if its properties deteriorate due to excessive deformation.

Many engineering applications require the use of moving grids. However, different problems require different solution methods. An important example is rotor-stator interaction which is common to turbomachinery and mixers: one part of the grid is attached to the stator and does not move, while another part is attached to the rotor and moves with it. The interface between the moving and fixed grids is usually a flat annulus. If grids match at the interface at the initial time, one can allow the rotating part of the grid to move while keeping the boundary points "glued" to the fixed grid, until the deformation becomes substantial (45° angles should be the maximum allowed); then, the boundary points "leap" one cell ahead and stay glued to the new location for a while. This kind of "clicking" grid has been used in these applications.

Another possibility is to let the moving grid "slide" along the interface without deformation. In this case the grids do not match at the interface, so some CVs have more neighbors than others. However, this situation is com-

pletely analogous to that encountered in block-structured grids with non-matching interfaces and can be handled by the method described in Sect. 8.6.5; the only difference is that the cell connectivity changes with time and has to be re-established after each time step. This approach is more flexible than the one described above; the grids can be of different kinds and/or fineness, and the interface can be an arbitrary surface. This approach can also be applied to flows around bodies passing each other, entering a tunnel, or moving in an enclosure with a known trajectory. Examples of such applications were presented by Lilek et al. (1997c) and Demirdžić et al. (1997).

The third approach is to use overlapping (Chimera) grids. Again, one grid is attached to the fixed part of the domain and the other to the moving body. This approach can be used even if the trajectory of the moving body is not known in advance, when it is very complicated, or when the surrounding domain is of a complex shape (e.g. when a sliding interface can not be constructed). The fixed grid may cover the whole "environment" in which the body is moving. The overlap region changes with time and the relationship between the grids needs to be re-established after each time step. Except for difficulties in ensuring exact conservation, there are almost no limitations on the applicability of this approach.

As noted above, the same equations and discretization methods apply to both the fixed and moving grids, the only difference being that on the fixed grid, the grid velocity $\boldsymbol{v}_{\text{b}}$ is obviously zero. Sometimes it may be advantageous to use different coordinate systems on the two domains; for example, one may use Cartesian velocity components in one part and polar components on the other grid. This is possible provided: (i) one adds the body forces due to frame acceleration and (ii) one transforms the vector components from one system to another at the interface or in the overlap region. Both of these are easy to do in principle but the programming may be tedious.

12.5 Free-Surface Flows

Flows with free surfaces are an especially difficult class of flows with moving boundaries. The position of the boundary is known only at the initial time; its location at later times has to be determined as part of the solution. To accomplish this, the SCL and the boundary conditions at the free surface must be used.

In the most common case, the free surface is an air-water boundary but other liquid-gas surfaces occur, as do liquid-liquid interfaces. If phase change at the free surface can be neglected, the following boundary conditions apply:

- The *kinematic condition* requires that the free surface be a sharp boundary separating the two fluids that allows no flow through it, i.e.:

$$[(\boldsymbol{v} - \boldsymbol{v}_{\text{b}}) \cdot \boldsymbol{n}]_{\text{fs}} = 0 \quad \text{or} \quad \dot{m}_{\text{fs}} = 0 \; , \tag{12.25}$$

where 'fs' denotes the free surface. This states that the normal component of the fluid velocity at the surface is the normal component of the velocity of the free surface, see Eq. (12.18).

- The *dynamic condition* requires that the forces acting on the fluid at the free surface be in equilibrium (momentum conservation at the free surface). This means that the normal forces on either side of the free surface are of equal magnitude and opposite direction, while the forces in the tangential direction are of equal magnitude and direction:

$$(\boldsymbol{n} \cdot \mathsf{T})_{\mathrm{l}} \cdot \boldsymbol{n} + \sigma K = -(\boldsymbol{n} \cdot \mathsf{T})_{\mathrm{g}} \cdot \boldsymbol{n} \,,$$
$$(\boldsymbol{n} \cdot \mathsf{T})_{\mathrm{l}} \cdot \boldsymbol{t} - \frac{\partial \sigma}{\partial t} = (\boldsymbol{n} \cdot \mathsf{T})_{\mathrm{g}} \cdot \boldsymbol{t} \,, \tag{12.26}$$
$$(\boldsymbol{n} \cdot \mathsf{T})_{\mathrm{l}} \cdot \boldsymbol{s} - \frac{\partial \sigma}{\partial s} = (\boldsymbol{n} \cdot \mathsf{T})_{\mathrm{g}} \cdot \boldsymbol{s} \,.$$

Here σ is the surface tension, $\boldsymbol{n}$, $\boldsymbol{t}$ and $\boldsymbol{s}$ are unit vectors in a local orthogonal coordinate system (n, t, s) at the free surface (n is the outward normal to the free surface while the other two lie in the tangent plane and are mutually orthogonal). The indices 'l' and 'g' denote liquid and gas, respectively, and K is the curvature of the free surface,

$$K = \frac{1}{R_t} + \frac{1}{R_s} \,, \tag{12.27}$$

with R_t and R_s being radii of curvature along coordinates t and s, see Fig. 12.4. The surface tension σ is the force per unit length of a surface element and acts tangential to the free surface; in Fig. 12.4, the magnitude of the force $\boldsymbol{f}_\sigma$ due to surface tension is $f_\sigma = \sigma\,\mathrm{d}l$. For an infinitesimally small surface element $\mathrm{d}S$, the tangential components of the surface tension forces cancel out when σ =const., and the normal component can be expressed as a local force that results in a pressure jump across the surface, as in Eq. (12.27).

Surface tension is a thermodynamic property of a liquid that depends on the temperature and other state variables such as chemical composition and surface cleanliness. If the temperature differences are small, the temperature dependence of σ can be linearized so that $\partial\sigma/\partial T$ is constant; it is usually negative. When the temperature varies substantially along the free surface, the gradient in surface tension results in a shear force that causes fluid to move from the hot region to the cold region. This phenomenon is called *Marangoni* or *capillary convection* and its importance is characterized by the dimensionless Marangoni number:

$$\mathrm{Ma} = -\frac{\partial \sigma}{\partial T} \frac{\Delta T\, L}{\mu \kappa} \,, \tag{12.28}$$

where ΔT is the bulk temperature difference across the domain, L is a characteristic length of the surface and κ is the thermal diffusivity.
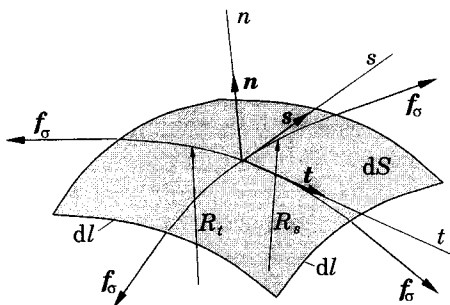
Fig. 12.4. On the description of boundary conditions at the free surface

In many applications the shear stress at the free surface can be neglected (e.g. ocean waves with no appreciable wind). The normal stress and the effect of the surface tension are also often neglected; in this case the dynamic boundary condition reduces to $p_l = p_g$.

The implementation of these boundary conditions is not as trivial as it would appear. If the position of the free surface were known, there would be little problem. The mass flux can be set to zero on cell faces lying on the free surface, and the forces acting on the cell face from outside can be calculated; if surface tension is neglected, only the pressure force remains. The problem is that the location of the free surface must be computed as part of the solution and is not usually known in advance. One can therefore directly implement only one of the boundary conditions at the free surface; the other must be used to locate the surface. Location of the surface must be done iteratively, greatly increasing the complexity of the task.

Many methods have been used to find the shape of the free surface. They can be classified into two major groups:

- Methods which treat the free surface as a sharp interface whose motion is followed *(interface-tracking methods)*. In this type of method, boundary-fitted grids are used and advanced each time the free surface is moved. In explicit methods, which must use small time steps, the problems associated with grid movement that were discussed earlier are often ignored.
- Methods which do not define the interface as a sharp boundary *(interface-capturing methods)*. The computation is performed on a fixed grid, which extends beyond the free surface. The shape of the free surface is determined by computing the fraction of each near-interface cell that is partially filled. This can be achieved by introducing massless particles at the free surface at the initial time and following their motion; this is called the *Marker-and-Cell* or MAC scheme that was proposed by Harlow and Welch (1965). Alternatively, one can solve a transport equation for the fraction of the cell

occupied by the liquid phase (the *Volume-of-Fluid* or VOF scheme, Hirt and Nichols, 1981).

There are also hybrid methods. All of these methods can also be applied to some kinds of two-phase flows as will be discussed in the following section.

Interface-Capturing Methods. The MAC scheme is attractive because it can treat complex phenomena like wave breaking. However, the computing effort is large, especially in three dimensions because, in addition to solving the equations governing fluid flow, one has to follow the motion of a large number of marker particles.

In the VOF method, in addition to the conservation equations for mass and momentum, one solves an equation for the filled fraction of each control volume, c so that $c = 1$ in filled CVs and $c = 0$ in empty CVs. From the continuity equation, one can show that the evolution of c is governed by the transport equation:

$$\frac{\partial c}{\partial t} + \operatorname{div}(c\boldsymbol{v}) = 0 . \tag{12.29}$$

In incompressible flows this equation is invariant with respect to interchange of c and $1 - c$; for this to be assured in the numerical method, mass conservation has to be strictly enforced.

The critical issue in this type of method is the discretization of convective term in Eq. (12.29). Low-order schemes (like the first-order accurate upwind method) smear the interface and introduce artificial mixing of the two fluids, so higher-order schemes are preferred. Since c must satisfy the condition

$$0 \leq c \leq 1 ,$$

it is important to ensure that the method does not generate overshoots or undershoots. Fortunately, it is possible to derive schemes which both keep the interface sharp and produce monotone profiles of c across it; see Leonard (1997) for some examples and Ubbink (1997) or Muzaferija and Perić (1999) for methods specifically designed for interface-capturing in free surface flows.

This approach is more efficient than the MAC scheme and can be applied to complex free surface shapes including breaking waves. However, the free surface profile is not sharply defined; it is usually smeared over one to three cells (similar to shocks in compressible flows). Local grid refinement is important for accurate resolution of the free surface. The refinement criterion is simple: cells with $0 < c < 1$ need to be refined. A method of this kind, called the marker and micro-cell method, has been developed by Raad and his colleagues (see, for example Chen et al., 1997).

There are several variants of the above approach. In the original VOF-method (Hirt and Nichols, 1981) Eq. (12.29) is solved in the whole domain to find the location of the free surface; the mass and momentum conservation equations are solved for the liquid phase only. The method can calculate flows

with overturning free surfaces, but the gas enclosed by the liquid phase will not feel buoyancy effects and will therefore behave in an unrealistic manner.

Kawamura and Miyata (1994) used Eq. 12.29) to calculate the distribution of the *density function* (the product of the density and the VOF) and to locate the free surface, which is the contour with $c = 0.5$. The computation of the motion of the liquid *and* gas flows are done separately. The free surface is treated as a boundary at which the kinematic and dynamic boundary conditions are applied. Cells which become irregular due to being cut by the free surface require special treatment (variable values are extrapolated to nodal locations lying on the other side of the interface). The method was used to calculate flows around ships and submerged bodies.

Alternatively, one can treat both fluids as a single fluid whose properties vary in space according to the volume fraction of each phase, i.e.:

$$\rho = \rho_1 c + \rho_2 (1 - c) \ , \quad \mu = \mu_1 c + \mu_2 (1 - c) \ , \tag{12.30}$$

where subscripts 1 and 2 denote the two fluids (e.g. liquid and gas). In this case, the interface is not treated as a boundary so no boundary conditions need to be prescribed on it. The interface is simply the location where the fluid properties change abruptly. However, solution of Eq. (12.29) implies that the kinematic condition is satisfied, and the dynamic condition is also implicitly taken into account. If surface tension is significant at the free surface, it can be taken into account by treating the force as a body force. Methods of this kind were presented by Brackbill et al. (1992), Lafaurie et al. (1994), Ubbink (1997), and Muzaferija and Perić (1999), among others. These methods can also deal with merging and fragmentation in multiphase flows.

The surface-tension force acts only in the region of interface, i.e. in partially filled cells, since in full or empty cells the gradient of c is zero:

$$\boldsymbol{F}_{\mathrm{fs}} = \int_{\Omega} \sigma \kappa \, \mathrm{grad}\, c \, \mathrm{d}\Omega \ . \tag{12.31}$$

However, there are problems when the surface-tension effects become dominant, like in the case of droplets or bubbles whose diameter is of the order of 1 mm or less and which move with very low velocity. In this case, there are two very large terms in the momentum equations (the pressure term and the body force representing the surface-tension effects) which have to balance each other; they are the only non-zero terms if the bubble or droplet is stationary. Due to the fact that curvature of the interface also depends on c,

$$\kappa = -\mathrm{div} \left(\frac{\mathrm{grad}\, c}{|\mathrm{grad}\, c|} \right) \ , \tag{12.32}$$

it is difficult to ensure on an arbitrary 3D grid that the two terms are identical, so their difference may cause the so called *parasitic currents*. These can be avoided by using special discretization methods in 2D (see Scardovelli and Zaleski, 1999, for some examples of such special methods); we do not know at

present of a method that eliminates the problem for unstructured arbitrary grids in 3D.

An example of the capabilities of interface-capturing methods is shown in Fig. 12.5, which illustrates the solution to the 'dam-break' problem, a standard test case for methods for computing free-surface flows. The barrier holding back the fluid is suddenly removed, leaving a free vertical water face. As water moves to the right along the floor, it hits an obstacle, flows over it and hits the opposite wall. The confined air escapes upwards as water falls to the floor on the other side of obstacle. Numerical results compare well with experiments by Koshizuka et al. (1995). The method has also been applied to vigorous sloshing in tanks, slamming of bodies onto liquid surface, and flow around ships and submerged bodies.
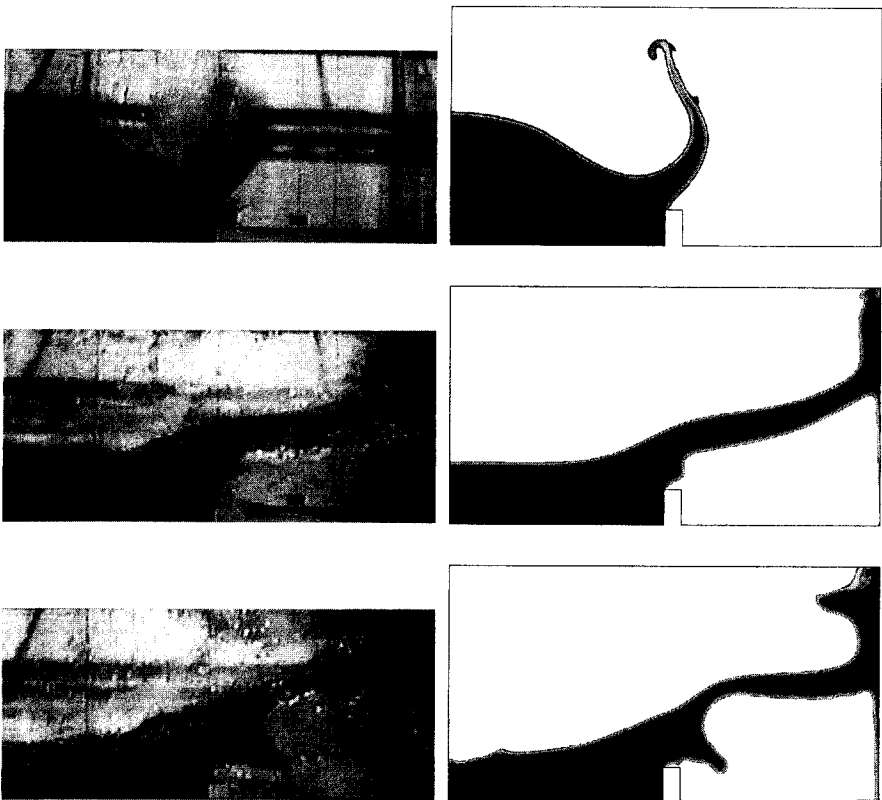
Fig. 12.5. Comparison of experimental visualization (left) and numerical prediction (right) of collapsing water column flow over an obstacle (experiments by Koshizuka et al., 1995; prediction by Muzaferija and Perić, 1999)

Another class of interface-capturing methods is based on the *level-set formulation*, introduced by Osher and Sethian (1988). The surface is defined as the one on which a level-set function $\phi = 0$. Other values of this function have no significance and to make it a smooth function, ϕ is typically initialized as the signed distance from the interface i.e. its value at any point is the distance from the nearest point on the surface and its sign is positive on one side and negative on the other. This function is then allowed to evolve as a solution of the transport equation:

$$\frac{\partial \phi}{\partial t} + \mathrm{div}\,(\phi \boldsymbol{v}) = 0 \,, \tag{12.33}$$

where v is the local fluid velocity and, at any time, the surface on which $\phi = 0$ is the interface. If the function ϕ becomes too complicated, it can be re-initialized in the manner described above. The advantage of this approach relative to the VOF-method is that ϕ varies smoothly across the interface while the volume fraction c is discontinuous there.

As in VOF-like methods, fluid properties are determined by the local value of ϕ but here, only the sign of ϕ is important.

When solving for the volume fraction c, its step-wise variation across the interface is usually not maintained—the step is smeared by the numerical approximation. As a result, the fluid properties experience a smooth change across the interface. In level-set methods, the step-wise variation of the properties is maintained, since we define

$$\rho = \rho_\mathrm{l} \quad \text{if} \quad \phi < 0 \,, \quad \rho = \rho_\mathrm{g} \quad \text{if} \quad \phi > 0 \,.$$

However, this usually causes problems when computing viscous flows so one needs to introduce a region of some finite thickness ϵ over which a smooth but rapid change of the properties occurs across the interface.

As noted above, the computed ϕ needs to be re-initialized every now and then. Sussman et al. (1994) proposed that this be done by solving the following equation:

$$\frac{\partial \phi}{\partial \tau} = \mathrm{sgn}(\phi_0)(1 - |\mathrm{grad}\,\phi|) \,, \tag{12.34}$$

until steady state is reached. This guarantees that ϕ has the same sign and zero level as ϕ_0 and fulfills the condition that $|\mathrm{grad}\,\phi| = 1$, making it similar to a signed distance function.

Since ϕ does not explicitly occur in any of the conservation equations, the original level-set method did not exactly conserve mass. Mass conservation can be enforced by making the right-hand side of Eq. (12.34) a function of the local mass imbalance $\Delta \dot{m}$ as was done by Zhang et al. (1998). The more frequently one solves this equation, the fewer iterations are needed to reach steady state; of course, frequent solution of this equation increases the computational cost so there is a trade-off.

Many level-set methods have been proposed; they differ in the choices for the various steps. Zhang et al. (1998) describe one such method, which they applied to bubble-merging and mold-filling, including melt solidification. They used a FV-method on structured, non-orthogonal grids to solve the conservation equations, and a FD-method for the level-set equation. An ENO-scheme was used to discretize the convective term in the latter.

Another version of this method is used to study flame propagation. In this case, the flame propagates relative to the fluid and this introduces the possibility that the surface will develop cusps, locations at which the surface normal is discontinuous. This will be discussed further below.

More details on level-set methods can be found in a book by Sethian (1996); see also Smiljanovski et al. (1997) and Reinecke et al. (1999) for examples of similar approaches used for flame tracking.

12.5.1 Interface-Tracking Methods

In the calculation of flows around submerged bodies, many authors linearize about the unperturbed free surface. This requires introduction of a *height function*, which is the free surface elevation relative to its unperturbed state:

$$z = H(x, y, t) . \tag{12.35}$$

The kinematic boundary condition (12.25) then becomes the following equation describing the local change of the height H:

$$\frac{\partial H}{\partial t} = u_z - u_x \frac{\partial H}{\partial x} - u_y \frac{\partial H}{\partial y} . \tag{12.36}$$

This equation can be integrated in time using the methods described in Chap. 6. The fluid velocity at the free surface is obtained either by extrapolation from the interior or by using the dynamic boundary condition (12.27).

This approach is usually used in conjunction with structured grids and explicit Euler time integration. Many authors use a FV method for the flow calculation and a FD method for the height equation and enforce both boundary conditions at the free surface only at the converged steady state (see e.g. Farmer et al., 1994).

Hino (1992) used a FV method with the enforcement of the SCL, thus satisfying all conditions at each time step and ensuring volume conservation. Similar methods were developed by Raithby et al. (1995), Thé et al. (1994) and Lilek (1995). One fully-conservative FV method of this type consists of the following steps:

- Solve the momentum equations using the specified pressure at the current free surface to obtain velocities u_i^*.
- Enforce local mass conservation in each CV by solving a pressure-correction equation, with a zero pressure correction boundary condition at the current free surface (see Sect. 10.2.2). Mass is conserved both globally and in each

CV, but the prescribed pressure at the free surface produces a velocity correction there, so that mass fluxes through the free surface are non-zero.

- Correct the position of the free surface to enforce the kinematic boundary condition. Each free-surface cell face is moved so that the volume flux due to its movement compensates the flux obtained in the previous step.
- Iterate until no further adjustment is necessary and both the continuity equation and momentum equations are satisfied.
- Advance to the next time step.

The critical issue for the efficiency and stability of the method is the algorithm for the movement of the free surface. The problem is that there is only one discrete equation per free-surface cell face but a larger number of grid nodes that have to be moved. Correct treatment of the intersections of the free surface with other boundaries (inlet, outlet, symmetry, walls) is essential if wave reflection and/or instability is to be avoided. We shall briefly describe one such method. Only two-time-level schemes are considered, but the approach can be extended.

The mass flux through a moving free-surface cell face is (see Eqs. (12.18) and (12.22)):

$$\dot{m}_{\mathrm{fs}} = \int_{S_{\mathrm{fs}}} \rho \boldsymbol{v} \cdot \boldsymbol{n} \,\mathrm{d}S - \int_{S_{\mathrm{fs}}} \boldsymbol{v}_{\mathrm{b}} \cdot \boldsymbol{n} \,\mathrm{d}S \approx \rho (\boldsymbol{v} \cdot \boldsymbol{n})_{\mathrm{fs}}^{\tau} S_{\mathrm{fs}}^{\tau} - \rho \dot{\Omega}_{\mathrm{fs}} \; . \tag{12.37}$$

The superscript τ denotes the time ($t_n < t_\tau < t_{n+1}$) at which the quantity is calculated; for the implicit Euler scheme, $t_\tau = t_{n+1}$, while for the Crank-Nicolson scheme, $t_\tau = frac12(t_n + t_{n+1})$.

The mass fluxes obtained from the pressure-correction equation with prescribed pressure at the free surface are non-zero; we compensate by displacing the free surface, i.e.:

$$\dot{m}_{\mathrm{fs}} + \rho \dot{\Omega}'_{\mathrm{fs}} = 0 \; . \tag{12.38}$$

From this equation we obtain the volume of fluid $\dot{\Omega}'_{\mathrm{fs}}$ which has to flow into or out of the CV due to free-surface motion. We need to obtain the coordinates of the CV vertices that lie on the free surface from this equation. This has to be done carefully and thus requires special attention. Because there is a single volumetric flow rate for each cell but a greater number of CV vertices, there are more unknowns than equations.

Thé et al. (1994) suggested using staggered CVs in the layer adjacent to free surface, but only in the continuity (pressure- correction) equation. The method was applied to several problems in 2D and showed good performance. However, it requires substantial adaptation of the solution method, especially in 3D; see Thé et al. (1994) for more details.

Another possibility is to define the CVs under the free surface not by vertices but by cell-face centers; the vertices are then defined by interpolating cell-face center locations, as shown in Fig. 12.6 for a 2D structured grid.
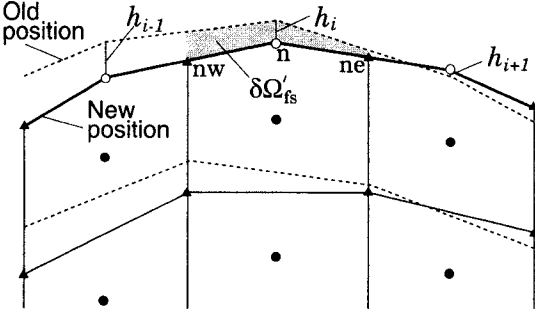
Fig. 12.6. Control volumes under the free surface, whose vertices lie on the free surface are defined by the coordinates of the cell face centers (open symbols); the volume swept by the cell face during the time step is shaded.

The volume swept by the free-surface cell face is then (see Eqs. (12.16) and (12.21)):

$$\delta \Omega'_{\mathrm{fs}} = \frac{1}{2} \Delta x \left(h_{\mathrm{nw}} + 2\, h_{\mathrm{n}} + h_{\mathrm{ne}} \right) , \tag{12.39}$$

where h is the distance the free surface-markers move during one time step; $h_{\mathrm{n}} = h_i$ while h_{nw} and h_{ne} are obtained by linearly interpolating h_i and h_{i-1} or h_i and h_{i+1}. By expressing h_{nw}, h_{n} and h_{ne} in terms of h_i, h_{i-1}, h_{i+1} and inserting the above expression into Eq. (12.38), we obtain a system of equations for the locations of the cell-face centers, h_i. In 2D, the system is tridiagonal and can be solved directly by the TDMA method of Sect. 5.2.3. In 3D, the system is block-tridiagonal and is best solved by one of the iterative solvers presented in Chap. 5. 'Boundary conditions' have to be specified at the CV vertices at the edges of the free surface. If the boundary is not allowed to move, $h = 0$. If the edge of the free surface is allowed to move, e.g. for an open system, the boundary condition should be of the non-reflective or 'wave-transmissive' type that does not cause wave reflection; the condition (9.4) is one appropriate possibility.

This approach was applied by Lilek (1995) to 2D and 3D problems on structured grids. When the lateral bounding surface has an irregular shape (e.g. a ship hull), the expressions for the volume $\delta \Omega'_{\mathrm{fs}}$ become complicated and require iterative solution on each outer iteration.

Muzaferija and Perić (1997) suggested a simpler approach. They noted that it is not necessary to calculate the swept volume from the geometry of the cell vertices on the free surface; it can be obtained from Eq. (12.38). The displacement of free-surface markers located above the cell-face center is defined by the height h , obtained from the known volume and cell-face area. The new vertex locations are then computed by interpolating h; the resulting swept volume is not exact and iterative correction is necessary. The method is suitable for implicit schemes, for which outer iterations are required at each time step anyway. The 'old' and 'new' locations shown in Fig. 12.6 are now the values for the current and preceding outer iterations; each outer iteration corrects the swept volume according to Eq. (12.38). At

the end of each time step, when the outer iterations converge, all corrections are zero. For a detailed description of this approach and its implementation on arbitrary unstructured 3D-grids, see Muzaferija and Perić (1997,1999).
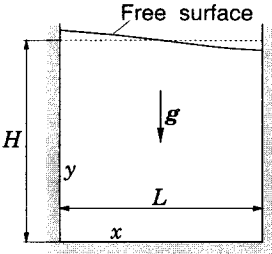
Fig. 12.7. Geometry and boundary conditions for the sloshing problem

Flows with free surfaces, like open channel flows, flows around ships etc. are characterized by the Froude number:

$$\mathrm{Fr} = \frac{v}{v_\mathrm{w}} = \frac{v}{\sqrt{gL}} , \tag{12.40}$$

where g is the gravity acceleration, v is the reference velocity, L is the reference length; $\sqrt{gL}$ is the velocity of a wave of length L in deep water. When $\mathrm{Fr} > 1$, the fluid velocity is greater than the wave speed and the flow is said to be *supercritical* and waves cannot travel upstream (as in supersonic compressible flow). When $\mathrm{Fr} < 1$, waves can travel in all directions. If the method of calculating the free surface shape is not properly implemented, disturbances in the form of small waves may be generated and it may not be possible to obtain a steady solution. A method which does not generate waves where they physically should not be (e.g. in front of ships) is said to satisfy the *radiation condition.*

We now present three examples of free-surface flow problems. The first is small-amplitude sloshing in a 2D tank. The initial free surface is a sine wave with amplitude 1% of the water depth; see Fig. refsloshgeo. If the liquid is inviscid, sloshing continues forever, i.e. the wave is not damped.

Figure 12.8 shows results of computations on a 40 × 40 CV uniform grid. The free-surface heights at the two side walls are shown as functions of time. In the inviscid case, (left figure) the amplitude is nearly unchanged after several oscillation periods. When viscosity is introduced (right figure), the wave amplitude decays with time and the period is changed. Linear theory, which is accurate for small amplitude waves, predicts the period to be $T = 3.55$. Lilek (1995) obtained $T = 3.55$ for the inviscid case and $T = 3.65$ for the viscous case.

The second example is the flow over a hydrofoil under a free surface. The geometry of the problem and the grid used are shown in Fig. 12.9. The grid is block-structured with non-matching interfaces and the steady-state shape of the free surface is shown. The hydrofoil of length $L = 0.203$
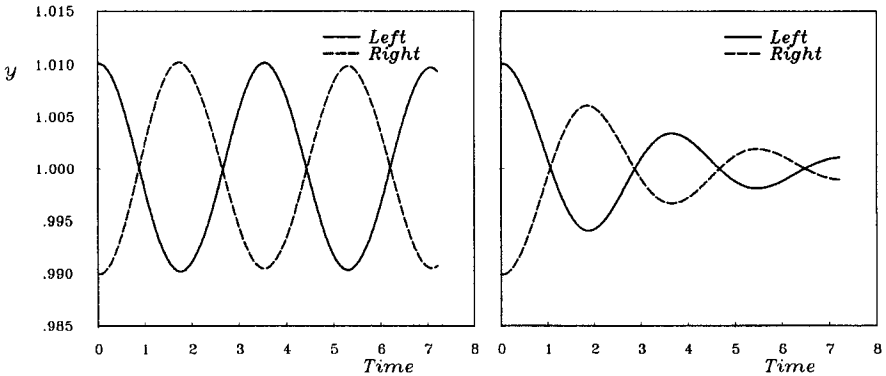
Fig. 12.8. The free surface heights at the side walls as functions of time for inviscid (left) and viscous (right) sloshing in a 2D tank (from Lilek, 1995)
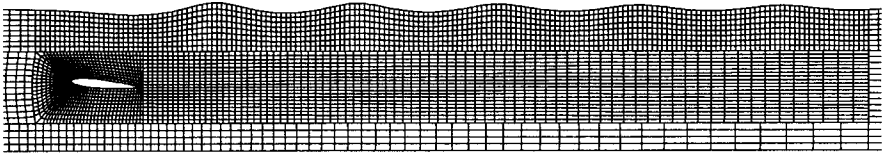
Fig. 12.9. A block-structured non-matching grid and the predicted free surface shape for the flow around a NACA 0012 hydrofoil at 5° angle of attack (Fr = 0.567)

m has a NACA 0012 profile and 5° angle of attack; the undisturbed water height above the profile is 0.21 m, and the Froude number is 0.567. This flow was studied experimentally by Duncan (1983). In Fig. 12.10 the steady-state free surface profiles calculated on four grids using CDS discretization are shown and compared with experimental data. The improvement with grid refinement is obvious and although the agreement with the experimental data is acceptable, it is clear that the converged result is not perfect. Since discretization errors are small (measured by the difference between solutions on the two finest grids), the discrepancy between prediction and experiment is due to modeling errors (different boundary conditions, turbulence effects etc.).

Finally we present results for 3D flow around the blunt-bow ship model shown in Fig. 12.11. The upper part is the waterline shape. The semi-circular bow with radius $R = 0.3$ m is followed by a parallel middle body 1 m long, and the 0.7 m long stern is defined by a spline. Thus the total length of the model is 2.0 m and the beam is 0.6 m. This shape is maintained 0.2 m above the load waterline to the deck and 0.3 m beneath the load waterline. There, a half body of revolution, obtained by rotating the waterline around its longitudinal axis completes the body which has a total draft of 0.6 m.
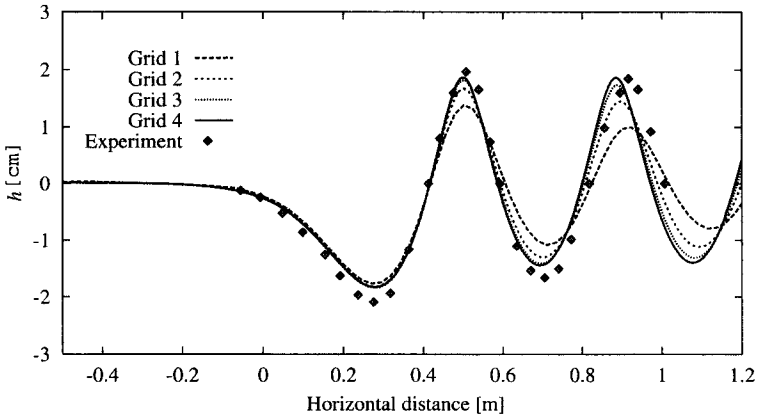
Fig. 12.10. The free-surface profile in flow around NACA 0012 foil at 5° angle of attack (Fr = 0.567), calculated on grids with 1 004 CV, 4 016 CV, 16 064 CV, and 64 256 CV, compared with experimental data of Duncan (1983) (from Yoo, 1998)

In both the experiments and computations, the model is held fixed, i.e. it is not allowed to sink and trim. The model moves at $v = 1.697$ m/s, corresponding to a Froude number Fr = 0.7, based on the hull draft. The Reynolds number, based on the hull length was around 3.4×10^6.

Since wave breaking occurs, interface-capturing method is used. The grid also extends 0.4 m into the air. Computations were carried out on three numerical grids. The coarsest one had 103950 CVs, the medium one had 411180 CVs, and the finest one had 2147628 CVs.
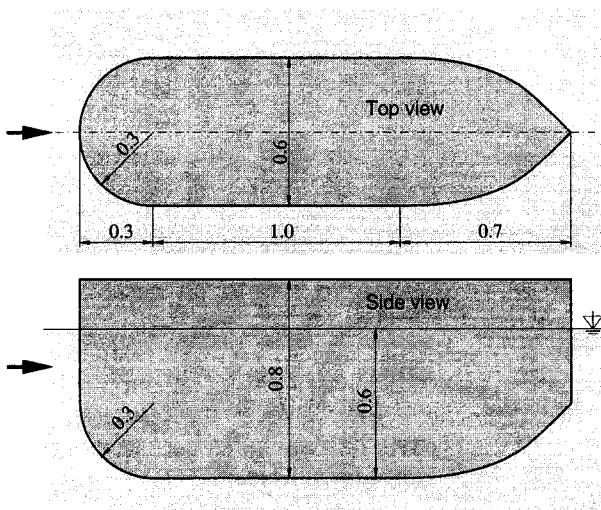
Fig. 12.11. Geometry of the hull model

The inlet, outlet, lateral, and bottom boundaries are placed about 1.5 model lengths away from the model. Due to the symmetry of the flow, only half of the model was considered. The grid resolution is high at the model surface and in the proximity of the waterline to better capture the boundary layer and free surface distortion. It expands continuously in all directions away from the hull and the waterline. The simple geometry of the model makes grid generation relatively easy. The coarsest grid is a matching block-structured grid, in which some cells are prisms; the finer grids were obtained by locally refining the base grid in the vicinity of the model and free surface.

At the inlet, the velocities of both water and air were set to the hull velocity, and the turbulence parameters were derived by assuming a turbulence intensity of 1% and a turbulent viscosity equal to the molecular viscosity. The top, bottom and lateral boundaries were treated as slip walls. At the symmetry plane, the symmetry boundary condition was enforced. At the outlet, extrapolation in streamwise direction was used, and the hydrodynamic pressure was specified according to a prescribed water level. The standard k-ϵ eddy-viscosity model was used.

The presence of wave breaking required that the simulation allow unsteady flow; over 12 periods of breaking were computed. Figure 12.12 shows time-averaged velocity vectors in the symmetry plane. The still waterline and average free-surface shape are also shown. Some back-flow is observed in the stern region; here, water flows upwards while air flows downwards, with free surface as a dividing streamline.
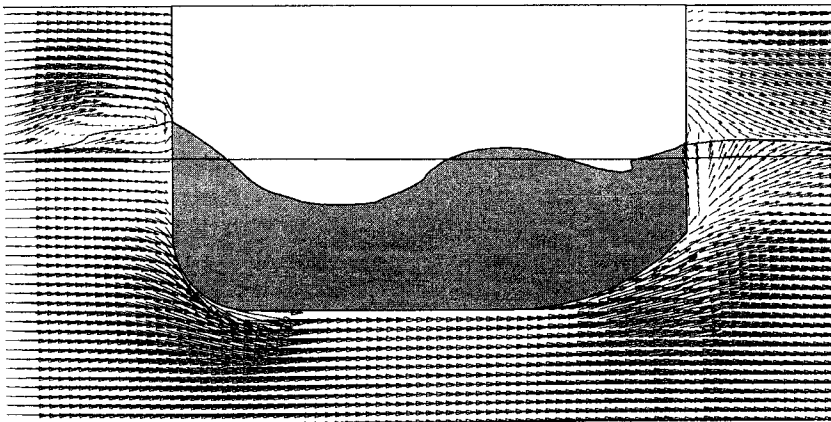
Fig. 12.12. Computed average velocity vectors, the undisturbed waterline, and the average free-surface profile in the symmetry plane (medium grid)

Figure 12.13 shows instantaneous experimental photographs of breaking waves around the model taken at the Ship Research Institute in Tokyo. They are compared with the instantaneous free surface obtained on the finest grid.

The similarity of the wave patterns is obvious. There is a breaking wave ahead of the bow followed by a deep trough at the hull shoulder, where the velocity is high and the pressure is low. The water rises steeply at mid-ship, where severe wave breaking takes place. The photograph shows a foamy region so a sharp interface (free surface) cannot be identified. In the simulation, interface is not sharp in this region (it is smeared over more cells than elsewhere), but the iso-surface $c = 0.5$ indicates the presence of wave breaking through its lack of smoothness.
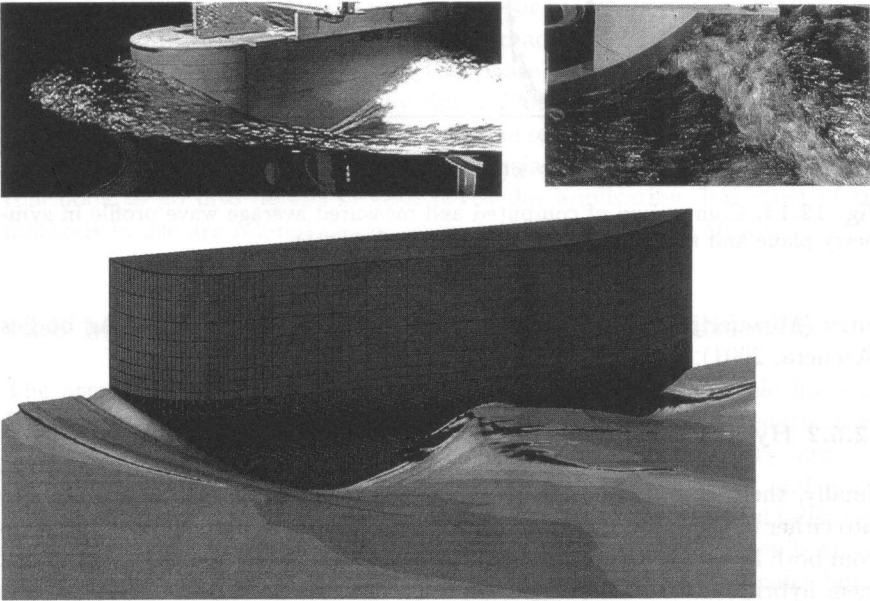
Fig. 12.13. Photographs of the instantaneous free surface (top left: bow and side; top right: stern; courtesy of Ship Research Institute, Tokyo) and the instantaneous shape of the free surface computed on the finest grid (bottom; from Azcueta et al., 2001)

Figure 12.14 compares the computed time-averaged free-surface elevation along the hull with experimental data from the Ship Research Institute, Tokyo. Surprisingly good agreement is observed between experiment and simulation; appreciable differences are only seen in the region of the breaking stern wave, which was so severe that measurements of wave height are uncertain. Also, the averaging period is probably not long enough. The computations were done without knowledge of the experimental results; for more details, see Azcueta et al. (2001).

This example demonstrates the versatility of the interface-capturing scheme. The same method has been used to study break-up of liquid jets in air (Albina et al., 2000), sloshing in tanks (Hadžić et al., 2001), water-
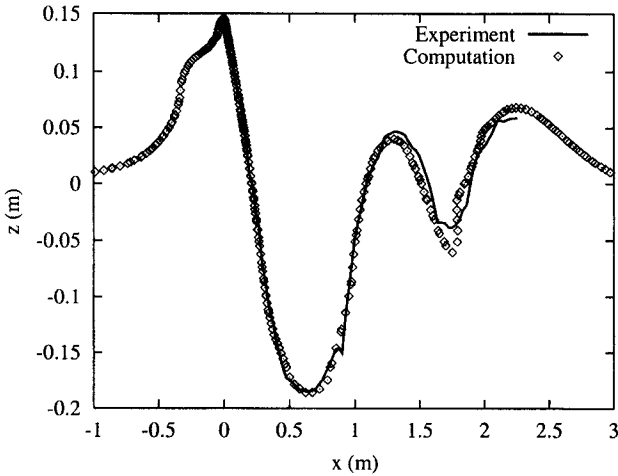
Fig. 12.14. Comparison of computed and measured average wave profile in symmetry plane and along hull; from Azcueta et al. (2001)

entry (Muzaferija et al., 1998), and flow around ships and floating bodies (Azcueta, 2001).

12.5.2 Hybrid Methods

Finally, there are methods for computing two-phase flows that do not fall into either of the categories described above. These methods borrow elements from both interface-capturing and interface-tracking methods so we shall call them hybrid methods. Among these are a method developed by Tryggvason and his colleagues that has been applied to bubbly flows, see Tryggvason and Unverdi (1990) and Bunner and Tryggvason (1999).

In this method, the fluid properties are smeared over a fixed number of grid points normal to the interface. The two phases are then treated as a single fluid with variable properties as in the interface-capturing methods. To keep the interface from becoming smeared, it is also tracked as in interface-tracking methods. This is done by moving marker particles using the velocity field generated by the flow solver. To maintain accuracy, marker particles are added or removed to maintain approximately equal spacing between them. The level-set method has been suggested as an alternative for this purpose. After each time step, the properties are re-computed.

Tryggvason and his colleagues have computed a number of flows with this method including some containing hundreds of bubbles of water vapor in water. Phase change, surface tension and merging and splitting of the bubbles can all be treated with this method.

Similar hybrid methods, in which both an additional equation for the volume fraction of one phase and the tracking of the interface are used, have been reported by Scardovelli and Zaleski et al (1999).

In many applications the interface between two fluids is not sharp; an example is a breaking wave or a hydraulic jump, where a region exists in which the water and air form a foamy mixture. In such a case it would be necessary to add a model for the mixing of the two fluids, similar to the models of turbulent transport in single-phase flows. Also, cavitation is an important phenomenon which falls into the class of two-phase flows which require hybrid methods for their simulation. The equation for the volume fraction of the gas phase has to have both a diffusion term (to account for turbulent diffusion) and a source term to account for the phase change; an example of such a method is the one presented by Sauer (2000). Special treatment is also required when computing bubble growth due to heat addition during boiling, as well as in some other flows with free surfaces; it is beyond the scope of this book to go into details of each particular application, but most of the methods in use are related to one of the methods described above.

12.6 Meteorological and Oceanographic Applications

The atmosphere and the oceans are the sites of the largest scale flows on Earth. The velocities may be tens of meters per second and the length scales are enormous so the Reynolds numbers are huge. Due to the very large aspect ratios of these flows (thousands of kilometers horizontally and a few kilometers of depth), the large-scale flow is almost two-dimensional (although vertical motions are important) while the flow at the small scales is three-dimensional. The Earth's rotation is a major force on the large scales but is less significant at the small scales. Stratification or a stable variation of density is important, mainly at the smaller scales. The forces and phenomena that play dominant roles are different on different scales.

Also, one needs predictions on different time scales. In the case of greatest interest to the public, one wants to predict the state of the atmosphere or ocean for a relatively short time in the future. In weather forecasting, the time scale is a few days while in the ocean, which changes more slowly, the scale is a few weeks to a few months. In either case, a method that is accurate in time is required. At the other extreme are climate studies, which require prediction of the average state of the atmosphere over a relatively long time period. In this case, the short term time behavior can be averaged out and the time accuracy requirement relaxed; however, it is essential to model the ocean as well as the atmosphere in this case. Because the actual state of the atmosphere or ocean is required, computations in these fields are nearly always large-eddy simulations.

Computations are done on a wide range of different length scales. The smallest region of interest is the atmospheric boundary layer or ocean mixed

layer that has dimensions of hundreds of meters. The next scale may be called the basin scale and consists of a city and its surroundings. On the regional- or meso-scale, one considers a domain that is a significant part of a continent or ocean. Finally there are the continental (or ocean) and global scales. In each case, computational resources dictate the number of grid points that can be used and thus the grid size. Phenomena that occur on smaller scales must be represented by an approximate model. Even on the smallest scales of interest, the size of the regions over which averaging must be done is obviously much larger than in engineering flows. Consequently, the models used to represent the smaller scales are much more important than in the engineering large-eddy simulations discussed in Chap. 9.

The fact that significant structures cannot be resolved in simulations at the largest scales requires that calculations be performed at a number of different scales; on each scale, the aim is to study phenomena particular to that scale. Meteorologists distinguish four to ten scales on which simulations are performed (depending on who does the counting). As one might expect, the literature on this subject is vast and we cannot even begin to cover all of what has been done.

As already noted, on the largest scales, atmospheric and oceanic flows are essentially two-dimensional (although there are important influences of vertical motion). In simulations of the global atmosphere or an entire ocean basin, the capacity of current computers requires the grid size in the horizontal directions be about a hundred kilometers. As a result, in these types of simulations, significant structures such as fronts (zones that exist between masses of fluid of different properties) have to be treated by approximate models to render their thickness sufficiently large that the grid can resolve them. Models of this kind are very difficult to construct and are a major source of error in predictions.

Three-dimensional motion is important only on the smallest scales of atmospheric of oceanographic motion. It is also important to note that, despite the high Reynolds numbers, only the portion of the atmosphere closest to the surface is turbulent; this is the atmospheric boundary layer and usually occupies a region about 1–3 km thick. Above the boundary layer, the atmosphere is stratified and remains laminar. Similarly, only the top layer of the ocean is turbulent; it is 100–300 m thick and is called the mixed layer. Modeling these layers is important because it is within them that the atmosphere and ocean interact and their impact on large-scale behavior is very important. These have been treated by large-eddy simulation methods similar to those described in Chap. 9.

The numerical methods used in these simulations vary somewhat with the scale on which the simulation is performed. For simulations at the smallest atmospheric scale, the boundary layer, one may use methods similar to those used in large-eddy simulation of engineering flows. For example, Coleman et al. (1992) used a spectral method originally designed for the engineering

boundary layer. Nieuwstadt et al. (1991) discuss results of a number of authors who used codes based on finite-volume methods. At the global scale, finite-volume methods are used but a spectral method specifically designed for the surface of a sphere is more common. This method uses spherical harmonics as the basis functions.

In choosing a time-advancement method, one must take into account the need for accuracy but it is also important to note that wave phenomena play a significant role in both meteorology and oceanography. The large weather systems that are familiar from weather maps and satellite photographs may be regarded as very large scale traveling waves. The numerical method must not amplify or dissipate them. For this reason, it is quite common to use the leapfrog method in these fields. This method is second-order accurate and neutrally stable for waves. Unfortunately, it is also unconditionally unstable (it amplifies exponentially decaying solutions) so it must be stabilized by restarting it approximately every ten time steps. There are a number of ways of doing this; one of the simplest is to use a different method for one time step.

12.7 Multiphase flows

Engineering applications often involve multiphase flows; examples are solid particles carried by gas or liquid flows (fluidized beds, dusty gases, and slurries), gas bubbles in liquid (bubbly fluids and boilers) or liquid droplets in gas (sprays), etc. A further complication is that multiphase flows often occur in combustion systems. In many combustors, liquid fuel or powdered coal is injected as a spray. In others, coal is burned in a fluidized bed.

The methods described in the previous section may be applied to some types of two-phase flows, especially those in which both phases are fluids. In these cases, the interface between the two fluids is treated explicitly as described above. Some of the methods were specifically designed for this type of flow. However, the computational cost associated with the treatment of interfaces limits these methods to flows in which the interfacial area is relatively small.

There are several other approaches to computing two-phase flows. The carrier or continuous phase fluid is always treated by an Eulerian approach, but the dispersed phase may be handled by either a Lagrangian or an Eulerian method.

The Lagrangian approach is often used when the mass loading of the dispersed phase is not very large and the particles of the dispersed phase are small; dusty gases and some fuel sprays are examples of flows to which this method might be applied. In this approach, the dispersed phase is represented by a finite number of particles or drops whose motion is computed in a Lagrangian manner. The number of particles whose motion is tracked is usually

much smaller than the actual number in the fluid. Each computational particle then represents a number (or packet) of actual particles; these are called packet methods. If phase change and combustion are not present and the loading is light, the effect of the dispersed phase on the carrier flow can be neglected and the latter can be computed first. Particles are then injected and their trajectories are computed using the pre-computed velocity field of the background fluid. (This approach is also used for flow visualization; one uses massless point particles and follows their motion to create streaklines.) This method requires interpolation of the velocity field to the particle location; the interpolation scheme needs to be at least as accurate as the methods used for time advancement. Accuracy also requires that the time step be chosen so that particles do not cross more than one cell in one time step.

When the mass loading of the dispersed phase is substantial, the influence of particles on the fluid motion has to be taken into account. If a packet method is used, the computation of particle trajectories and fluid flow must be done simultaneously and iteration is needed; each particle contributes momentum (and energy and mass) to the gas in the cell in which it is located. Interaction between particles (collision, agglomeration, and splitting) and between particles and walls needs to be modeled. For these exchanges, correlations based on experiment have been used but the uncertainties may be rather large. These issues require another book to be described in any detail; see the newly published work by Crowe et al. (1998) for a description of most widely used methods.

For large mass loadings and when phase change takes place, an Eulerian approach (the *two-fluid model*) is applied to both phases. In this case, both phases are treated as continua with separate velocity and temperature fields; the two phases interact via exchange terms analogous to those used in the mixed Eulerian-Lagrangian approach. A function defines how much of each cell is occupied by each phase. The principles of two-fluid models are described in detail by Ishii (1975); see also Crowe et al. (1998) for a description of some methods for gas-particle and gas-droplet flows. The methods used to compute these flows are similar to those described earlier in this book, except for the addition of the interaction terms and boundary conditions and, of course, twice as many equations need to be solved.

12.8 Combustion

Another important problem area deals with flows in which combustion i.e. chemical reaction with significant heat release plays an important role. Some of the applications should be obvious to the reader. Some combustors operate at nearly constant pressure so the principal effect of heat release is reduction of the density. In many combustion systems it is not unusual for the absolute temperature to increase by a factor of five to eight through the flame; the density decreases by the same factor. In such a case, there is no possibility

that the density differences can be dealt with by means of the Boussinesq approximation discussed earlier. In other systems (engine cylinders are the most common example) there are large changes in both pressure and density.

It is possible to do direct numerical simulation of turbulent combusting flows but only for very simple flows. It is important to note that the speed of travel of a flame relative to the gas is rarely greater than 1 m/s (explosions or detonations are an exception). This speed is much lower than the speed of sound in the gas; usually, the fluid velocities are also well below the sound speed. Then the Mach number is much less than unity and we have the strange situation of a flow with large temperature and density changes that is essentially incompressible.

It is possible to compute combusting flows by solving the compressible equations of motion. This has been done (see Poinsot et al., 1991). The problem is that, as we have pointed out earlier, most methods designed for compressible flows become very inefficient when applied to low speed flows, raising the cost of a simulation. For this reason, these simulations are very expensive; this is especially so when the chemistry is simple. However, when more realistic (and therefore more complex) chemistry is included, the range of time scales associated with the chemical reaction is almost always very large and this dictates that small time steps be used. In other words, the equations are very stiff. In this case, the penalty for using compressible flow methods may be largely eliminated.

An alternative approach is to introduce a low Mach number approximation (McMurtry et al., 1986). One starts with the equations describing a compressible flow and assumes that all of the quantities to be computed may be expressed as power series in the Mach number. This is a non- singular perturbation theory so no special care is required. The results are, however, somewhat surprising. To lowest (zeroth) order, the momentum equations reduce to the statement that the pressure $p^{(0)}$ is constant everywhere. This is the thermodynamic pressure and the density and temperature of the gas are related by its equation of state. The continuity equation has the compressible (variable density) form, which is no surprise. At the next order, the momentum equations in their usual form are recovered but they contain only the gradient of the first-order pressure, $p^{(1)}$, which is essentially the dynamic head found in the incompressible equations. These equations resemble the incompressible Navier-Stokes equations and can be solved by methods that have been given in this book.

In the theory of combustion, two idealized cases are distinguished. In the first, the reactants are completely mixed before any reaction takes place and we have the case of premixed flames. Internal combustion engines are close to this limit. In premixed combustion, the reaction zone or flame propagates relative to the fluid at the laminar flame speed. In the other case, the reactants mix and react at the same time and one speaks of non-premixed combustion. These two cases are quite different and are treated separately. Of course,

there are many situations that are not close to either limit; they are called partially premixed. For a complete treatment of the theory of combustion, the reader is advised to consult the well-known work by Williams (1985).

The key parameter in reacting flows is the ratio of the flow time scale to the chemical time scale; it is known as the Damköhler number, Da. When the Damköhler number is very large, chemical reaction is so fast that it takes place almost instantaneously after the reactants have mixed. In this limit, flames are very thin and the flow is said to be mixing-dominated. Indeed, if the effects of heat release can be ignored, it is possible to treat the limit $Da \to \infty$ as one involving a passive scalar and the methods discussed at the beginning of this chapter can be used.

For the calculation of practical combustors, in which the flow is almost always turbulent, it is necessary to rely on solution of the Reynolds-averaged Navier-Stokes (RANS) equations. This approach and the turbulence models that need to be used in conjunction with it for non-reacting flows were described in Chapter 9. When combustion is present, it is necessary to solve additional equations that describe the concentrations of the reacting species and to include models that allow one to compute the reaction rate. We shall describe some of these models in the remainder of this section.

The most obvious approach, that of Reynolds-averaging the equations for a reacting flow does not work. The reason is that chemical reaction rates are very strong functions of temperature. For example, the reaction rate between species A and species B might be given by:

$$R_{AB} = K\mathrm{e}^{-E_a/RT} Y_A Y_B \ , \tag{12.41}$$

where E_a is called the activation energy, R is the gas constant, and Y_A and Y_B are the concentrations of the two species. The presence of the Aarhenius factor $\mathrm{e}^{(-E_a/RT)}$ is what makes the problem difficult. It varies so rapidly with temperature that replacing T with its Reynolds-averaged value produces large errors.

In a high Damköhler number non-premixed turbulent flame, the reaction takes place in thin wrinkled flame zones. For this case, several approaches have been used, of which we will briefly describe two. Despite the significant difference in philosophy and appearance between them, they are more alike than it would seem.

In the first approach, one takes the point of view that, since mixing is the slower process, the reaction rate is determined by how fast it takes place. In that case, the rate of reaction between two species A and B is given by an expression of the form:

$$R_{AB} = \frac{Y_A Y_B}{\tau} \ , \tag{12.42}$$

where τ is the time scale for mixing. For example, if the k-ε model is used, $\tau = k/\varepsilon$. In the k-ω model, $\tau = 1/\omega$. A number of models of this kind have been proposed, perhaps the best known of which is the eddy break-up model

of Spalding (1978). Models of this type are in common use for the prediction of the performance of industrial furnaces.

Another type of model for non-premixed combustion is the laminar flamelet model. Under stagnant conditions, a non-premixed flame would slowly decay as its thickness increases with time. To prevent this from happening, there must be a compressive strain on the flame. The state of the flame is determined by this strain rate or, its more commonly used surrogate, the rate of scalar dissipation, χ. Then it is assumed that the local structure of a flame is determined by just a few parameters; at minimum, one needs the local concentrations of the reactants and the scalar dissipation rate. The data on flame structure is tabulated. Then the volumetric reaction rate is computed as the product of the reaction rate obtained by table look-up and the flame area per unit volume. A number of versions of the equation for the flame area have been given. We shall not present one here but it should suffice to say that these models contain terms describing the increase of the flame area by stretching of the flame and destruction of flame area.

For premixed flames, which propagate relative to the flow, the equivalent of the flamelet model is a kind of level-set method. If the flame is assumed to be the location where some variable $G = 0$, then G satisfies the equation:

$$\frac{\partial G}{\partial t} + u_j \frac{\partial G}{\partial x_j} = S_L |\nabla G| \,, \tag{12.43}$$

where S_L is the laminar flame speed. One can show that the rate of consumption of reactants is $S_L|\nabla G|$, thus completing the model. In more complex versions of the model, the flame speed may be a function of the local strain rate just as it depends on the scalar dissipation rate in non-premixed flames.

Finally, note that there are many effects that are very difficult to include in any combustion model. Among these are ignition (the initiation of a flame) and extinction (the destruction of a flame). Models for turbulent combustion are undergoing rapid development at the present time and no snapshot of the field can remain current for very long. The reader interested in this subject should consult the recent book by Peters (2000).

A. Appendices

A.1 List of Computer Codes and How to Access Them

A number of computer codes embodying some of the methods described in this book can be obtained by readers via the Internet. These codes may be useful as they stand, but they can also serve as the starting point for further development. They will be updated from time to time, and new codes may be added; for a current list of codes, download the `read.me` file in the directory given below.

All computer codes can be accessed using `ftp` from the publisher's server `ftp.springer.de`; the procedure is as follows:

- Type `ftp ftp.springer.de` on your computer;
- Type `ftp` when prompted for a user name;
- Type *your E-mail address* when prompted for a password;
- Type `cd pub/technik/peric` to access the main directory;
- Type `get read.me` to obtain a copy of the `read.me` file; read this file and proceed further as explained in it to obtain codes you desire.

Included are codes used to solve the one- and two-dimensional generic conservation equation; these were used to do the examples in Chaps. 3, 4 and 6. Several schemes for discretization of the convective and diffusive terms and time integration are used in these codes. They can be used to study features of the schemes, including convergence and discretization errors and the relative efficiency of the solvers. They can also be used as the basis for student assignments; they could, for example, be asked to modify the discretization scheme and/or boundary conditions.

Several solvers are given in the initial package including:

- TDMA solver for 1D problems;
- Line-by-line TDMA solver for 2D problems (five-point molecule);
- Stone's ILU solver (SIP) for 2D and 3D problems (five- and seven-point molecules; the 3D version is also given in vectorized form);
- Conjugate gradient solver preconditioned by the Incomplete Cholesky method (ICCG) for symmetric matrices in 2- and 3-D (five- and seven-point molecules);
- A modified SIP solver for a nine-point molecule in 2D;

- CGSTAB solver for non-symmetric matrices and 3D problems;
- Multigrid solver for 2D problems using Gauss-Seidel, SIP, and ICCG as smoothers.

Other codes may be added in the future.

Finally, there are several codes for solving fluid flow and heat transfer problems. The source codes of the following are included:

- A code for generating Cartesian 2D grids;
- A code for generating non-orthogonal structured 2D grids;
- A code for post-processing 2D data on Cartesian and non-orthogonal grids, which can plot the grid, velocity vectors, profiles of any quantity on lines of $x =$ const. or $y =$ const., and contours of any quantity in black and white or color (the output is a postscript file);
- A FV code on a Cartesian 2D grid with the staggered variable arrangement, for steady problems;
- A FV code using Cartesian 2D grids with the colocated variable arrangement, for steady or unsteady problems; versions for serial and parallel computers using the PVM are provided;
- A FV code using Cartesian 3D grids and the colocated variable arrangement, for steady and unsteady problems, with multigrid applied to the outer iterations;
- A FV code using boundary-fitted non-orthogonal 2D grids and the colocated variable arrangement, for laminar steady or unsteady flows (including moving grids);
- Versions of the above code that include k–ε and k–ω turbulence models with wall functions and a version that does not use wall functions;
- A multigrid version of the above code for laminar flows (multigrid applied to outer iterations).

The codes are programmed in standard FORTRAN77 and have been tested on many computers. For the larger codes there are also explanation files in the directory; many comment lines are included in each code, including suggestions how to adapt them to 3D problems on unstructured grids.

In addition to the source codes, the directory contains a subdirectory `Comet`, which contains a version of the commercial code **Comet**(*Continuum mechanics engineering tool*) provided by ICCM (Institute of Computational Continuum Mechanics) GmbH, Hamburg. It is a full-featured code; the only difference compared with the version that is sold lies in the limit to the maximum number of CVs that can be used and the exclusion of user coding. Included are versions for the Linux and Windows NT operating systems. Documentation in pdf-format explains how to use the pre-processor, flow solver, and post-processor; several sub-directories contain sets of examples of application, including predictions of inviscid steady flows, creeping flows, laminar steady flows, laminar unsteady flows, turbulent flows, compressible subsonic, transonic, and supersonic flows, flows with free surfaces, flows with moving

boundaries etc. These examples also demonstrate the method of estimating discretization errors, checking and improving of grid quality, error-guided local mesh refinement etc. Suggestions are also given for parameter variation, further testing, and other suitable test cases that can be used both to learn fluid dynamics and to demonstrate the use of CFD-tools for solving problems in engineering practice. A quick guide is also given for setting-up grids and parameters for flow computations in simple geometry without the need to read all of the documentation. We believe that this CFD-package will be especially useful to those who teach a CFD-course, since they will be able to present applications of CFD on-line without lengthy preparations. A set of postscript-files with lecture transparences that the authors used in their courses and also some presentation files for PowerPoint and StarOffice programs are included for this purpose.

There is also a directory that contains color plots of different quantities from various flow predictions, as well as mpeg-animations showing simulations of unsteady flows. These materials, which could not be printed in the book are a useful supplement to a course on CFD. We hope that the readers will share our view.

Finally, the directory contains a file named `errata`; in it, errors which might be found will be documented (we hope that this file will be very small, if not empty). Postscript files of modified pages compared to previous editions of this book are also included in a separate directory; this will enable buyers of the older editions to obtain free updates.

A.2 List of Frequently Used Abbreviations

1D	one-dimensional
2D	two-dimensional
3D	three-dimensional
ADI	alternating direction implicit
BDS	backward difference scheme
CDS	central difference scheme
CFD	computational fluid dynamics
CG	conjugate gradient method
CGSTAB	CG stabilized
CM	control mass
CV	control volume
CVFEM	control-volume-based finite element method
DNS	direct numerical simulation
ENO	essentially non-oscillatory
FAS	full approximation scheme
FD	finite difference
FDS	forward difference scheme
FE	finite elements

FFT	fast Fourier transform
FMG	full multigrid method
FV	finite volume
GC	global communication
GS	Gauss-Seidel method
ICCG	CG preconditioned by incomplete Cholesky method
ILU	incomplete lower-upper decomposition
LC	local communication
LES	large eddy simulation
LU	lower-upper decomposition
MAC	marker-and-cell
MG	multigrid
ODE	ordinary differential equation
PDE	partial differential equation
PVM	parallel virtual machine
RANS	Reynolds averaged Navier-Stokes
SCL	space conservation law
SGS	subgrid scale
SIP	strongly implicit procedure
SOR	successive over-relaxation
TDMA	tridiagonal matrix algorithm
TVD	total variation diminishing
UDS	upwind difference scheme
VOF	volume-of-fluid

References

1. Abgrall, R. (1994): On essentially non-oscillatory schemes on unstructured meshes: Analysis and implementation. J. Copmput. Phys., **114**, 45
2. Albina, F.-O., Muzaferija, S., Perić, M. (2000): Numerical simulation of jet instabilities. Proc. 16th Annual Conference on Liquid Atomization and Spray Systems, Darmstadt, VI.1.1–VI.1.6
3. Anderson, D.A, Tannehill J.C., Pletcher, R.H. (1984): Computational fluid mechanics and heat transfer. Hemisphere, New York
4. Arcilla, A.S., Häuser, J., Eiseman, P.R., Thompson, J.F. (eds.) (1991): Numerical grid generation in computational fluid dynamics and related fields. North-Holland, Amsterdam
5. Aris, R. (1989): Vectors, tensors and the basic equations of fluid mechanics. Dover Publications, New York
6. Azcueta, R. (2001): Computation of turbulent free-surface flows around ships and floating bodies. PhD Thesis, Technical University of Hamburg-Harburg, Germany
7. Azcueta, R., Muzaferija, S., Perić, M. (2001): Numerical simulation of flow around blunt bow model. Proc. Workshop on Numerical Simulation of Two-Phase Flows, Ship Research Institute, Tokyo, 27–37
8. Baker, A.J. (1983): Finite element computational fluid mechanics. McGraw-Hill, New York
9. Baliga, R.B., Patankar, S.V. (1983): A control-volume finite element method for two-dimensional fluid flow and heat transfer. Numer. Heat Transfer, **6**, 245–261
10. Baliga, R.B. (1997): Control-volume finite element method for fluid flow and heat transfer. In W.J. Minkowycz, E.M. Sparrow (eds.), Advances in Numerical Heat Transfer, chap. 3, 97–135, Taylor and Francis, New York
11. Bardina, J., Ferziger, J.H., Reynolds, W.C. (1980): Improved subgrid models for large eddy simulation. AIAA paper 80–1357
12. Bastian, P., Horton, G. (1989): Parallelization of robust multi-grid methods: ILU factorization and frequency decomposition method. In W. Hackbusch, R. Rannacher (eds.), Notes on Numerical Fluid Mechanics, **30**, 24–36, Vieweg, Braunschweig
13. Beam, R.M., Warming, R.F. (1978): An implicit factored scheme for the compressible Navier-Stokes equations. AIAA J., **16**, 393–402
14. Berger, M.J., Oliger, J. (1984): Adaptive mesh refinement for hyperbolic partial differential equations. J. Copmput. Phys., **53**, 484
15. Bertram, V., Jensen, G. (1994): Recent applications of computational fluid dynamics. Ship Technology Research, **44**, 131–134
16. Bewley, T., Moin, P., Temam, R. (1994): Optimal control of turbulent channel flows. In Active Control of Vibration and Noise, 221–227, Amer. Soc. Mech. Eng., Design Eng. Div. DE v 75 1994. ASME, New York

17. Bird, R.B., Stewart, W.E., Lightfoot, E.N. (1962): Transport phenomena. Wiley, New York
18. Boris, J.P., Book, D.L. (1973): Flux-corrected transport. I.SHASTA, a fluid transport algorithm that works. J. Copmput. Phys., **11**, 38–69
19. Brackbill, J.U., Kothe, D.B., Zemaach, C., (1992): A continuum method for modeling surface tension. J. Comput. Phys., **100**, 335–354
20. Bradshaw, P., Launder, B.E., J.L. Lumley,J.L. (1994): Collaborative testing of turbulence models. In K.N. Ghia, U. Ghia, D. Goldstein (eds.), Advances in Computational Fluid Mechanics, ASME FED, **196**, ASME, New York
21. Brandt, A. (1984): Multigrid techniques: 1984 guide with applications to fluid dynamics. GMD-Studien Nr. 85, Gesellschaft für Mathematik und Datenverarbeitung (GMD), Bonn, Germany
22. Briggs, D.R., Ferziger, J.H., Koseff, J.R., Monismith, S.G. (1996): Entrainment in a shear free mixing layer. J. Fluid Mech., **310**, 215–241
23. Bückle, U., Perić, M. (1992): Numerical simulation of buoyant and thermocapillary convection in a square cavity. Numer. Heat Transfer, Part A (Applications), **21**, 101–121
24. Bunner, B., Tryggvason, G. (1999): Direct numerical simulations of three-dimensional bubbly flows. Phys. Fluids, **11**, 1967–1969
25. Burmeister, J., Horton, G. (1991): Time-parallel solution of the Navier-Stokes equations. Proc. 3rd European Multigrid Conference, Birkhäuser Verlag, Basel
26. Cain, A.B., Reynolds, W.C., Ferziger, J.H. (1981): A three-dimensional simulation of transition and early turbulence in a time-developing mixing layer. Report TF–14, Dept. Mech. Engrg., Stanford University
27. Canuto, C., Hussaini, M.Y., Quarteroni, A., Zang, T.A. (1987): Spectral methods in fluid mechanics. Springer, Berlin
28. Caretto, L.S., Gosman, A.D., Patankar, S.V., Spalding, D.B. (1972): Two calculation procedures for steady, three-dimensional flows with recirculation. Proc. Third Int. Conf. Numer. Methods Fluid Dyn., Paris
29. Caruso, S.C., Ferziger, J.H., Oliger, J. (1985): An adaptive grid method for incompressible flows. Report TF–23, Dept. Mech. Engrg., Stanford University
30. Cebeci, T., Bradshaw, P. (1984): Physical and computational aspects of convective heat transfer. Springer, New York
31. Chen, S., Johnson, D.B., Raad, P.E., Fadda, D. (1997): The surface marker and micro-cell method. Intl. J. Num. Methods Fluids, **25**, 749–778
32. Choi, H., Moin, P., Kim, J. (1994): Active turbulence control for drag reduction in wall-bounded flows. J. Fluid Mech., **262**, 75–110
33. Choi, H., Moin, P. (1994): Effects of the computational time step on numerical solutions of turbulent flow. J. Comput. Phys., **113**, 1–4
34. Chorin, A.J. (1967): A numerical method for solving incompressible viscous flow problems. J. Copmput. Phys., **2**, 12
35. Chung, T.J. (1978): Finite element analysis in fluid dynamics. McGraw-Hill, New York
36. Coelho, P., Pereira, J.C.F., Carvalho, M.G. (1991): Calculation of laminar recirculating flows using a local non-staggered grid refinement system. Int. J. Numer. Methods Fluids, **12**, 535–557
37. Coleman, G.N., Ferziger, J.H., Spalart, P.R. (1992): Direct simulation of the stably stratified turbulent Eckman layer. J. Fluid Mech., **244**, 667
38. Cooley, J.W., Tukey, J.W. (1965): An algorithm for the machine calculation of complex Fourier series. Math. Comput., **19**, 297–301
39. Craft, T.J., Launder, B.E. (1995): Improvements in near-wall Reynolds stress modelling for complex flow geometries. Proc. 10th Symp. Turbulent Shear Flows, Pen. State Univ., August 1995

40. Craft, T.J., Launder, B.E., Suga, K. (1995): A non-linear eddy viscosity model including sensitivity to stress anisotropy. Proc. 10th Symp. Turbulent Shear Flows, Pen. State Univ., August 1995
41. Crowe, C., Sommerfeld, M., Tsuji, Y. (1998): Multiphase flows with droplets and particles. CRC Press, Boca Raton, Florida
42. Demirdžić, I., Perić, M. (1988): Space conservation law in finite volume calculations of fluid flow. Int. J. Numer. Methods Fluids, **8**, 1037–1050
43. Demirdžić, I., Perić, M. (1990): Finite volume method for prediction of fluid flow in arbitrarily shaped domains with moving boundaries. Int. J. Numer. Methods Fluids, **10**, 771–790
44. Demirdžić, I., Lilek, Ž., Perić, M. (1993): A colocated finite volume method for predicting flows at all speeds. Int. J. Numer. Methods Fluids, **16**, 1029–1050
45. Demirdžić, I., Muzaferija, S. (1994): Finite volume method for stress analysis in complex domains. Int. J. Numer. Methods Engrg., **37**, 3751–3766
46. Demirdžić, I., Muzaferija S. (1995): Numerical method for coupled fluid flow, heat transfer and stress analysis using unstructured moving meshes with cells of arbitrary topology. Comput. Methods Appl. Mech. Engrg., **125**, 235–255
47. Demirdžić, I., Muzaferija, S., Perić, M., Schreck, E. (1997): Numerical method for simulation of flow problems involving moving and sliding grids. Proc. 7th Int. Symp. Computational Fluid Dynamics, Int. Academic Publishers, Beijing, 359–364
48. Demmel, J.W., Heath, M.T., van der Vorst, H.A. (1993): Parallel numerical linear algebra. In Acta Numerica, **2**, 111–197, Cambridge Univ. Press, New York
49. Deng, G.B., Piquet, J., Queutey, P., Visonneau, M. (1994): Incompressible flow calculations with a consistent physical interpolation finite volume approach. Computers Fluids, **23**, 1029–1047
50. Domaradzki J.A., Saiki E.M. (1997): A subgrid-scale model based on the estimation of unresolved scales of turbulence. Phys. Fluids, **9**, 2148–2164
51. Drazin, P. G., Reid, W.H. (1981): Hydrodynamic stability. Cambridge Univ. Press, Cambridge
52. Duncan, J.H. (1983): The breaking and non-breaking wave resistance of a two-dimensional hydrofoil. J. Fluid Mech., **126**, 507–520
53. Durbin, P.A. (1991): Near-wall turbulence closure modeling without 'damping functions'. Theoret. Comput. Fluid Dynamics, **3**, 1–13
54. Durbin, P.A., Pettersson Reif, B.A. (2001): Statistical theory and modeling for turbulent flows. Wiley, Chichester, England
55. Durst, F., Kadinskii, L., Perić, M., Schäfer, M. (1992): Numerical study of transport phenomena in MOCVD reactors using a finite volume multigrid solver. J. Crystal Growth, **125**, 612–626
56. Farmer, J., Martinelli, L., Jameson, A. (1994): Fast multigrid method for solving incompressible hydrodynamic problems with free surfaces. AIAA J., **32**, 1175–1182
57. Ferziger, J.H. (1987): Simulation of turbulent incompressible flows. J. Copmput. Phys., **69**, 1–48
58. Ferziger, J.H. (1993): Estimation and reduction of numerical error. Presented at ASME Winter Annual Meeting, Washington
59. Ferziger, J.H. (1995): Large eddy simulation. In M.Y. Hussaini, T. Gatski (eds.), Simulaiton and Modeling of Turbulent Flows, Cambridge Univ. Press, New York
60. Ferziger, J.H. (1998): Numerical methods for engineering application. Wiley, New York
61. Ferziger, J.H., Perić, M. (1996): Further discussion of numerical errors in CFD. Int. J. Numer. Methods Fluids, **23**, 1–12

62. Fletcher, R. (1976): Conjugate gradient methods for indefinite systems. Lecture Notes in Mathematics, **506**, 773–789
63. Fletcher, C.A.J. (1991): Computational techniques for fluid dynamics, vol. I. Springer, Berlin
64. Fox, R.W., McDonald, A.T. (1982): Introduction to fluid mechanics. Wiley, New York
65. Friedrich, O. (1998): Weighted essentially non-oscillatory schemes for the interpolation of mean values on unstructured grids. J. Copmput. Phys., **144**, 194–212
66. Germano, M., Piomelli, U., Moin, P., Cabot, W.H. (1990): A dynamic subgrid scale eddy viscosity model. Proc. Summer Workshop, Center for Turbulence Research, Stanford CA
67. Galpin, P.F., Raithby, G.D. (1986): Numerical solution of problems in incompressible fluid flow: tratment of the temperature-velocity coupling. Numer. Heat Transfer, **10**, 105–129
68. Ghia, U., Ghia, K.N., Shin, C.T. (1982): High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method. J. Copmput. Phys., **48**, 387–411
69. Girault, V., Raviart, P.-A. (1986): Finite element methods for Navier-Stokes equations. Springer, Berlin
70. Golub, G.H., van Loan, C. (1990): Matrix computations. Johns Hopkins Univ. Press, Baltimore
71. Gosman, A.D. (1984): Prediction of in-cylinder processes in reciprocating internal combustion engines. In R. Glowinski, J.-L. Lions (eds.), Computing Methods in Applied Sciences and Engineering, 609–629, Elsevier (North-Holland), Amsterdam
72. Gresho, P.M., Sani, R.L. (1990): On pressure boundary conditions for the incompressible Navier-Stokes equations. Int. J. Numer. Methods Fluids, **7**, 11–46
73. Hackbusch, W. (1984): Parabolic multi-grid methods, in R. Glowinski, J.-R. Lions (eds.), Computing Methods in Applied Sciences and Engineering, North Holland, Amsterdam
74. Hackbusch, W. (1985): Multi-grid methods and applications. Springer, Berlin
75. Hackbusch, W., Trottenberg, U. (eds.) (1991): Proc. Third European Multigrid Conference. International Series of Numerical Mathematics, Birkhäuser, Basel
76. Hadžić, I. (1999): Second-moment closure modelling of transitional and unsteady turbulent flows. PhD Thesis, Delft University of Technology
77. Hadžić, I., Mallon, F., Perić, M. (2001): Numerical simulation of sloshing. Proc. Workshop on Numerical Simulation of Two-Phase Flows, Ship Research Institute, Tokyo, 45–57
78. Hageman, L.A., Young, D.M. (1981): Applied iterative methods. Wiley, New York
79. Hanjalić, K., Launder, B.E. (1976): Contribution towards a Reynolds-stress closure for low Reynolds number turbulence. J. Fluid Mech., **74**, 593–610
80. Hanjalić, K., Launder, B.E. (1980): Sensitizing the dissiptaion equation to irrotational strains. J. Fluids Engrg., **102**, 34–40
81. Hanjalić, K. (1994): Advanced turbulence closure models: Review of current status and future prospects. Int. J. Heat Fluid Flow, **15**, 178–203
82. Harlow, F.H., Welsh, J.E. (1965): Numerical calculation of time dependent viscous incompressible flow with free surface. Phys. Fluids, **8**, 2182–2189
83. Harrison, R.J. (1991): Portable tools and applications for parallel computers. Int. J. Quantum Chem., **40**, 847–863

84. Hinatsu, M., Ferziger, J.H. (1991): Numerical computation of unsteady incompressible flow in complex geometry using a composite multigrid technique. Int. J. Numer. Methods Fluids, **13**, 971–997
85. Hino, T. (1992): Computation of viscous flows with free surface around an advancing ship/ Proc. 2nd Osaka Int. Colloquium on Viscous Fluid Dynamics in Ship and Ocean Technology, Osaka Univ.
86. Hirsch, C. (1991): Numerical computation of internal and external flows, vol. I & II. Wiley, New York
87. Hirt, C.W., Amsden, A.A., Cook, J.L. (1974): An arbitrary Lagrangean-Eulerian computing method for all flow speeds. J. Copmput. Phys., **14**, 227
88. Hirt, C.W., Nicholls, B.D. (1981): Volume of fluid (VOF) method for dynamics of free boundaries. J. Comput. Phys., **39**, 201–221
89. Holt, S.E., Koseff J.R., Ferziger, J.H. (1992): A numerical study of the evolution and structure of homogeneous stably stratified sheared turbulence. J. Fluid Mech., **237**, 499–539
90. Hortmann, M., Perić, M., Scheuerer, G. (1990): Finite volume multigrid prediction of laminar natural convection: bench-mark solutions. Int. J. Numer. Methods Fluids, **11**, 189–207
91. Horton, G. (1991): Ein zeitparalleles Lösungsverfahren für die Navier-Stokes-Gleichungen. Dissertation, Universität Erlangen-Nürnberg
92. Hsu, C. (1991): A curvilinear-coordinate method for momentum, heat and mass transfer in domains of irregular geometry. PhD Thesis, University of Minnesota
93. Hubbard, B.J., Chen, H.C. (1994): A Chimera scheme for incompressible viscous flows with applications to submarine hydrodynamics. AIAA Paper 94–2210
94. Hubbard, B.J., Chen, H.C. (1995): Calculations of unsteady flows around bodies with relative motion using a Chimera RANS method. Proc. 10th ASCE Engineering Mechanics Conference, vol II, 782–785, Univ. of Colorado at Boulder, Boulder, CO, May 21–24
95. Hutchinson, B.R., Raithby, G.D. (1986): A multigrid method based on the additive correction strategy. Numer. Heat Transfer, **9**, 511–537
96. Hutchinson, B.R., Galpin, P.F., Raithby, G.D., (1988): Application of additive correction multigrid to the coupled fluid flow equations. Numer. Heat Transfer, **13**, 133–147
97. Ishii, M. (1975): Thermo-fluid dynamic theory of two-phase flow. Eyrolles, Paris
98. Isaacson, E., Keller, H.B. (1966): Analysis of numerical methods. Wiley, New York
99. Issa, R.I. (1986): Solution of implicitly discretized fluid flow equations by operator-splitting. J. Copmput. Phys., **62**, 40–65
100. Issa, R.I., Lockwood, F.C. (1977): On the prediction of two-dimensional supersonic viscous interaction near walls. AIAA J., **15**, 182–188
101. ITTC (1983): Cooperative experiments on Wigley parabolic models in Japan. 17th ITTC Resistance Committee Report, 2nd ed.
102. Ivey, G.N., Imberger, J. (1991): On the nature of turbulence in a stratified fluid, Part I. The energetics of mixing. J. Phys. Oceanogr., **21**, 650–660
103. Jones, W.P. (1994): Turbulence modelling and numerical solution methods for variable density and combusting flows. In P.A. Libby, F.A. Williams (eds.), Turbulent Reacting Flows, 309–374, Academic Press, London
104. Kadinski, L., Perić, M. (1996): Numerical study of grey-body surface radiation coupled with fluid flow for general geometries using a finite volume multigrid solver. Int. J. Numer. Meth. Heat Fluid Flow, **6**, 3–18
105. Karki, K.C., Patankar, S.V. (1989): Pressure based calculation procedure for viscous flows at all speeds in arbitrary configurations. AIAA J., **27**, 1167–1174

106. Katapodes, F.V., Street, R.L., Ferziger, J.H. (2000): Subfilter scale scalar transport for large eddy simulations. Amer. Meteorological Soc. Conf. on Ocean Simulation, June 2000
107. Kawamura, T., Miyata, H. (1994): Simulation of nonlinear ship flows by density-function method. J. Soc. Naval Architects Japan, **176**, 1–10
108. Kays, W.M., Crawford, M.E. (1978): Convective heat and mass transfer. McGraw-Hill, New York
109. Kenjereš, S. (1998): Numerical modelling of complex buoyancy-driven flows. PhD Thesis, Delft University of Technology
110. Kim, J., Moin, P. (1985): Application of a fractional step method to incompressible Navier-Stokes equations. J. Copmput. Phys., **59**, 308–323
111. Kim, J., Moin, P., Moser, R.D. (1987): Turbulence statistics in fully developed channel flow at low Reynolds number. J. Fluid Mech., **177**, 133–166
112. Khosla, P.K., Rubin, S.G. (1974): A diagonally dominant second-order accurate implicit scheme. Computers Fluids, **2**, 207–209
113. Kordula, W., Vinokur, M. (1983): Efficient computation of volume in flow predictions. AIAA J., **21**, 917–918
114. Koshizuka, S., Tamako, H., Oka, Y. (1995): A particle method for incompressible viscous flow with fluid fragmentation. Computational Fluid Dynamics J., **4**, 29–46
115. Kwak, D., Chang, J.L.C., Shanks, S.P., Chakravarthy, S.R. (1986): A three-dimensional incompressible Navier-Stokes flow solver using primitive variables. AIAA J., **24**, 390–396
116. Lafaurie, B., Nardone, C., Scardovelli, R., Zaleski, S., Zanetti, G. (1994): Modelling merging and fragmentation in multiphase flows with SURFER. J. Comput. Phys., **113**, 134–147
117. Launder, B.E. (1989): Second moment closure: Present ... and future? Int. J. Heat Fluid Flow, **10**, 282–300
118. Launder, B.E. (1990): Whither turbulence? Turbulence at the crossroads. In J.L. Lumley (ed.), Lecture Notes in Physics, **357**, 439–485, Springer, Berlin
119. Launder, B.E., Li, S.P. (1994): On the elimination of wall topography parameters from second moment closure. Phys. Fluids, **6**, 999–1006
120. Leister, H.-J., Perić, M. (1992): Numerical simulation of a 3D Chochralski melt flow by a finite volume multigrid algorithm. J. Crystal Growth, **123**, 567–574
121. Leister, H.-J., Perić, M. (1994).: Vectorized strongly implicit solving procedure for seven-diagonal coefficient matrix. Int. J. Numer. Meth. Heat Fluid Flow, **4**, 159–172
122. Leonard, A. (1974): Energy cascade in large eddy simulations of turbulent fluid flows. Adv. Geophys., **18A**, 237
123. Leonard, A., Wray, A.A. (1982): A new numerical method for the simulation of three dimensional flow in a pipe. In E. Krause (ed.), Lecture Notes in Physics, **170**, Springer, Berlin
124. Leonard, A. (1995): Direct numerical simulation. In T. Gatski (ed.), Turbulence and its Simulation, Springer, New York
125. Leonard, B.P. (1979): A stable and accurate convection modelling procedure based on quadratic upstream interpolation. Copmput. Meth. Appl. Mech. Engrg., **19**, 59–98
126. Leonard, B.P. (1997): Bounded higher-order upwind multidimensional finite-volume convection-diffusion algorithms. In W.J. Minkowycz, E.M. Sparrow (eds.), Advances in Numerical Heat Transfer, chap. 1, 1–57, Taylor and Francis, New York
127. Leschziner, M.A. (1989): Modelling turbulent recirculating flows by finite-volume methods. Int. J. Heat Fluid Flow, **10**, 186–202

128. Lilek, Ž., Nadarajah, S., Perić, M., Tindal, M.J., Yianneskis, M. (1991): Measurement and simulation of the flow around a poppet valve. Proc. 8th Symp. Turbulent Shear Flows, 13.2.1–13.2.6, TU München, Sept. 9–11
129. Lilek, Ž., Perić, M. (1995): A fourth-order finite volume method with colocated variable arrangement. Computers Fluids, **24**, 239–252
130. Lilek, Ž., Schreck, E., Perić, M. (1995): Parallelization of implicit methods for flow simulation. In S.G. Wagner (ed.), Notes on Numerical Fluid Mechanics, **50**, 135–146, Vieweg, Braunschweig
131. Lilek, Ž. (1995): Ein Finite-Volumen Verfahren zur Berechnung von inkompressiblen und kompressiblen Strömungen in komplexen Geometrien mit beweglichen Rändern und freien Oberflächen. Dissertation, University of Hamburg, Germany
132. Lilek, Ž., Muzaferija, S., Perić, M. (1997a): Efficiency and accuracy aspects of a full-multigrid SIMPLE algorithm for three-dimensional flows. Numer. Heat Transfer, Part B, **31**, 23–42
133. Lilek, Ž., Muzaferija, S., Perić, M., Seidl, V. (1997b): An implicit finite-volume method using non-matching blocks of structured grid. Numer. Heat Transfer, Part B, **32**, 385–401
134. Lilek, Ž., Muzaferija, S., Perić, M., Seidl, V. (1997c): Computation of unsteady flows using non-matching blocks of structured grid. Numer. Heat Transfer, Part B, **32**, 369–384
135. Liu, X.-D., Osher, S., Chan, T. (1994): Weighted essentially non-oscillatory schemes. J. Comput. Phys., **115**, 200
136. Loh K.C., Domaradzki J.A. (1999): The subgrid-scale estimation model on non-uniform grids. Phys. Fluids, **11**, 3786–3792
137. MacCormack, R.W. (1969): The effect of viscosity in hypervelocity impact cratering. AIAA-Paper 60-354
138. Majumdar, S., Rodi W., Zhu J. (1992): Three-dimensional finite-volume method for incompressible flows with complex boundaries. ASME J. Fluids Engrg., **114**, 496–503
139. Maliska, C.R., Raithby, G.D. (1984): A method for computing three-dimensional flows using non-orthogonal boundary-fitted coordinates. Int. J. Numer. Methods Fluids, **4**, 518–537
140. Manhart, M., Wengle, H. (1994): Large eddy simulation of turbulent boundary layer over a hemisphere. In P. Voke, L. Kleiser, J.P. Chollet (eds.), Proc. 1st ERCOFTAC Workshop on Direct and Large Eddy Simulation, 299–310, Kluwer Academic Publishers, Dordrecht
141. Marchuk, G.M. (1975): Methods of numerical mathematics. Springer, Berlin
142. Mason, M.L., Putnam, L.E., Re, R.J. (1980): The effect of throat contouring on two-dimensional converging-diverging nozzle at static conditions. NASA Techn. Paper No. 1704
143. Masson, C., Saabas, H.J., Baliga, R.B. (1994): Co-located equal-order control-volume finite element method for two-dimensional axisymmetric incompressible fluid flow. Int. J. Numer. Methods Fluids, **18**, 1–26
144. McCormick, S.F. (ed.) (1987): Multigrid methods. Society for Industrial and Applied Mathematics (SIAM), Philadelphia
145. McMillan, O.J., Ferziger, J.H. (1980): Tests of new subgrid scale models in strained turbulence. AIAA-Paper 80-1339
146. McMurtry, P.A., Jou, W.H., Riley, J.J., Metcalfe, R.W. (1986): Direct numerical simulations of a reacting mixing layer with chemical heat release. AIAA J., **24**, 962–970
147. Mellor, G.L., Yamada, T. (1982): Development of a turbulence closure model for geophysical fluid problems. Rev. Geophysics, **20**, 851–875

148. Meneveau, C., Lund, T.S., Cabot, W.H. (1996): A Lagrangian dynamic subgrid-scale model of turbulence. J. Fluid Mech., **319**, 353–385
149. Menter, F.R. (1993): Zonal two-equations k-ω turbulence models for aerodynamic flows. AIAA-Paper 93-2906
150. Moser, R.D., Moin, P., Leonard, A. (1983): A spectral numerical method for the Navier-Stokes equations with applications to Taylor-Couette flow. J. Comput. Phys., **52**, 524–544
151. Muzaferija, S. (1994): Adaptive finite volume method for flow predictions using unstructured meshes and multigrid approach. PhD Thesis, University of London
152. Muzaferija, S., Perić, M., Seidl, V. (1995): Computation of flow around circular cylinder in a channel. Internal Report, Institut für Schiffbau, University of Hamburg
153. Muzaferija, S., Perić, M. (1997): Computation of free-surface flows using finite volume method and moving grids. Numer. Heat Transfer, Part B, **32**, 369–384
154. Muzaferija, S., Gosman, A.D. (1997): Finite-volume CFD procedure and adaptive error control strategy for grids of arbitrary topology. J. Comput. Physics, **138**, 766–787
155. Muzaferija, S., Perić, M., Sames, P.C., Shellin, T. (1998): A two-fluid Navier-Stokes solver to simulate water entry. Proc. 22nd Symposium on Naval Hydrodynamics, Washington, D.C.
156. Muzaferija, S., Perić, M. (1999): Computation of free surface flows using interface-tracking and interface-capturing methods. In O. Mahrenholtz, M. Markiewicz (eds.), Nonlinear Water Wave Interaction, Chap. 2, 59–100, WIT Press, Southampton
157. Nieuwstadt, F.T.M., Mason, P.J., Moeng, C.-H., Schuman, U. (1991): Large-eddy simulation of the convective boundary layer: A comparison of four computer codes. In F. Durst et al. (eds.), Turbulent Shear Flows, **8**, Springer, Berlin
158. Oden, J.T. (1972): Finite elements of non-linear continua. McGraw-Hill, New York
159. Oden, J.T., Strouboulis, T., Devloo, P. (1986): Adaptive finite element methods for the analysis of inviscid compressible flow: Part I. Fast refinement/unrefinement and moving mesh methods for unstructured meshes. Copmput. Meth. Appl. Mech. Engrg., **59**, 327–362
160. Oden, J.T., Demkowitz, L., Rachowitz, W., Westerman, T.A. (1989): Toward a universal h-p adaptive finite element strategy: Part 2. A posteriori error estimation. Comput. Meth. Appl. Mech. Engrg., **77**, 113–180
161. Osher, S., Sethian, J.A. (1988): Fronts propagating with curvature-dependent speed: algorithms based on Hamilton-Jacobi formulations. J. Comput. Phys., **79**, 12–49
162. Patankar, S.V. (1980): Numerical heat transfer and fluid flow. McGraw-Hill, New York
163. Patankar, S.V., Spalding, D.B. (1977): Genmix: A general computer program for two-dimensional parabolic phenomena. Pergamon Press, Oxford
164. Patel, V.C., Rodi, W., Scheuerer, G. (1985): Turbulence models for near-wall and law-Reynolds number flows: a review. AIAA J., **23**, 1308–1319
165. Perić, M. (1987).: Efficient semi-implicit solving algorithm for nine-diagonal coefficient matrix. Numer. Heat Transfer, **11**, 251–279
166. Perić, M. (1990): Analysis of pressure-velocity coupling on non-orthogonal grids. Numerical Heat Transfer, Part B (Fundamentals), **17**, 63–82
167. Perić, M. (1993): Natural convection in trapezoidal cavities. Num. Heat Transfer, Part A (Applications), **24**, 213–219

168. Perić, M., Kessler, R., Scheuerer, G. (1988): Comparison of finite volume numerical methods with staggered and colocated grids. Computers Fluids, **16**, 389–403
169. Perić, M., Rüger, M., Scheuerer, G. (1989): A finite volume multigrid method for calculating turbulent flows. Proc. 7th Symposium on Turbulent Shear Flows, vol. 1., pp. 7.3.1–7.3.6, Stanford University
170. Perić, M., Schäfer, M., Schreck, E. (1993): Numerical simulation of complex fluid flows on MIMD computers. In R.B. Pelz et al. (eds.), Parallel Computational Fluid Dynamics '92, Elsevier, Amsterdam
171. Perić, M., Schreck, E. (1995): Analysis of efficiency of implicit CFD methods on MIMD computers. Proc. Parallel CFD '95 Conference, Pasadena, June 1995.
172. Perng, C.Y., Street, R.L. (1991): A coupled multigrid–domain-splitting technique for simulating icompressible flows in geometrically complex domains. Int. J. Numer. Methods Fluids, **13**, 269–286
173. Peters, N. (1998): The use of flamelet models in CFD-simulations. ERCOFTAC Bulletin, **38**, 71–78
174. Peters, N. (2000) Turbulent Combustion. Cambridge U. Press, Cambridge
175. Piomelli, U., Ferziger, J.H., Moin, P., Kim, J. (1989): New approximate boundary conditions for large eddy simulations of wall-bounded flows. Phys. Fluids, **A1**, 1061–1068
176. Poinsot, T., Veynante, D., Candel, S. (1991): Quenching processes and premixed turbulent combustion diagrams. J. Fluid Mech., **228**, 561–605
177. Prakash, C. (1981): A finite element method for predicting flow through ducts with arbitrary cross section. PhD Thesis, University of Minnesota
178. Press, W.H., Flannery, B.P., Teukolsky, S.A., Vettering, W.T. (1987): Numerical recipes. Cambridge Univ. Press, Cambridge
179. Raithby, G.D. (1976): Skew upstream differencing schemes for problems involving fluid flow. Copmput. Meth. Appl. Mech. Engrg., **9**, 153–164
180. Raithby, G.D., Schneider, G.E. (1979): Numerical solution of problems in incompressible fluid flow: treatment of the velocity-pressure coupling. Numer. Heat Transfer, **2**, 417–440
181. Raithby, G.D., Xu, W.-X., Stubley, G.D. (1995): Prediction of incompressible free surface flows with an element-based finite volume method. Comput. Fluid Dynamics J., **4**, 353–371
182. Raw, M.J. (1995): A coupled algebraic multigrid method for the 3D Navier-Stokes equations. In W. Hackbusch, G. Wittum (eds.), Fast Solvers for Flow Problems, Notes on Numerical Fluid Mechanics, **49**, 204–215, Vieweg, Braunschweig
183. Reinecke, M., Hillebrandt, W., Niemeyer, J.C., Klein, R., Grobl, A. (1999): A new model for deflagration fronts in reactive fluids. Astronomy and Astrophysics, **347**, 724–733
184. Rhie, C.M., Chow, W.L. (1983): A numerical study of the turbulent flow past an isolated airfoil with trailing edge separation. AIAA J., **21**, 1525–1532
185. Richardson, L.F. (1910): The approximate arithmetical solution by finite differences of physical problems involving differential equations with an application to the stresses in a masonry dam. Trans. Roy. Soc. London, Ser. A, **210**, 307–357
186. Richtmyer, R.D., Morton, K.W. (1967): Difference methods for initial value problems. Wiley, New York
187. Rizzi, A., Viviand, H. (eds.) (1981): Numerical methods for the computation of inviscid transonic flows with shock waves. Notes on Numerical Fluid mechanics, **3**, Vieweg, Braunschweig
188. Roache, P.J. (1994): Perspective: a method for uniform reporting of grid refinement studies. ASME J. Fluids Engrg., **116**, 405–413

189. Rogallo, R.S. (1981): Numerical experiments in homogeneous turbulence. NASA Tech. Memo 81315
190. Rodi, W., Bonnin, J.-C., Buchal, T. (eds.) (1995): Proc. ERCOFTAC Workshop on Data Bases and Testing of Caluclation Methods for Turbulent Flows, April 3–7, Univ. Karlsruhe, Germany
191. Saad, Y., Schultz, M.H. (1986): GMRES: a generalized residual algorithm for solving non-symmetric linear systems. SIAM J. Sci. Stat. Comput., **7**, 856–869
192. Scardovelli, R., Zaleski, S. (1999): Direct numerical simulation of free-surface and interfacial flow. Annu. Rev. Fluid Mech., **31**, 567–603
193. Schneider, G.E., Zedan, M. (1981): A modified strongly implicit procedure for the numerical solution of field problems. Numer. Heat Transfer, **4**, 1–19
194. Schneider, G.E., Raw, M.J. (1987): Control-volume finite-element method for heat transfer and fluid flow using colocated variables— 1. Computational procedure. Numer. Heat Transfer, **11**, 363–390
195. Schreck, E., Perić, M. (1993): Computation of fluid flow with a parallel multigrid solver. Int. J. Numer. Methods Fluids, **16**, 303–327
196. Sedov, L.I. (1971): A course in continuum mechanics, vol. 1. Walters-Noordhoft Publishing, Groningen
197. Seidl, V., Perić, M., Schmidt, S. (1995): Space- and time-parallel Navier-Stokes solver for 3D block-adaptive Cartesian grids. Proc. Parallel CFD '95 Conference, Pasadena, June 1995.
198. Seidl, V. (1997): Entwicklung und Anwendung eines parallelen Finite-Volumen-Verfahrens zur Strömungssimulation auf unstrukturierten Gittern mit lokaler Verfeinerung. Dissertation, University of Hamburg, Germany
199. Sethian, J.A. (1996): Level set methods. Cambridge University Press, Cambridge, UK
200. Shah, K.B., Ferziger, J.H. (1997): A fluid mechanicians view of wind engineering: large eddy simulation of flow over a cubical obstacle. In R.N. Meroney, B. Bienkiewicz (eds.), Computational Wind Engineering, **2**, 211–226, Elsevier, Amsterdam
201. Shih L.H., Koseff J.R., Ferziger J.H., Rehmann C.R. (2000): Scaling and parameterization of stratified homogeneous turbulent shear flow. J. Fluid Mech., **412**, 1–20
202. Slattery, J.C. (1972): Momentum, energy and mass transfer in continua. McGraw-Hill, New York
203. Smagorinsky, J. (1963): General circulation experiments with the primitive equations, part I: the basic experiment. Monthly Weather Rev., **91**, 99–164
204. Smiljanovski, V., Moser, V., Klein R. (1997): A capturing-tracking hybrid scheme for deflagration discontinuities. Combustion Theory and Modelling, **1**, 183–215
205. Sonar, Th. (1997): On the construction of essentially non-oscillatory finite volume approximations to hyperbolic conservation laws on general triangulations: Polynomial recovery, accuracy and stencil selection. Comput. Methods Appl. Mech. Engrg., **140**, 157
206. Sonneveld, P. (1989): CGS, a fast Lanczos type solver for non-symmetric linear systems. SIAM J. Sci. Stat. Copmput., **10**, 36–52
207. Spalding, D.B. (1972): A novel finite-difference formulation for differential expressions involving both first and second derivatives. Int. J. Numer. Methods Engrg., **4**, 551–559
208. Spalding D.B. (1978): General theory of turbulent combustion. J. Energy, **2**, 16–23

209. Steger, J.L., Warming, R.F. (1981): Flux vector splitting of the inviscid gas-dynamic equations with applications to finite difference methods. J. Comput. Phys., **40**, 263–293
210. Stoker, J.J. (1957): Water waves. Interscience, New York
211. Stone, H.L. (1968): Iterative solution of implicit approximations of multidimensional partial differential equations. SIAM J. Numer. Anal., **5**, 530–558
212. Strikwerda, J.C. (1983): Finite difference methods for the incompressible Navier-Stoker equations — A survey. MRC Tech. Summary Rept. 2584, Math. Res. Ctr., University of Wisconsin
213. Sunderam, V.S. (1990): PVM: a framework for parallel distributed computing. Cocurrency: Practice and Experience, **2**, 315–339
214. Sussman, M., Smereka, P., Osher, S. (1994): A level set approach for computing solutions to incompressible two-phase flow. J. Comput. Phys., **114**, 146–159
215. Tennekes, H., Lumley, J.L. (1976): A first course in turbulence. MIT Press
216. Thé, J.L., Raithby, G.D., Stubley, G.D. (1994): Surface-adaptive finite-volume method for solving free-surface flows. Numer. Heat Transfer, Part B, **26**, 367–380
217. Thomas, P.D., Lombard, C.K. (1979): Geometric conservation law and its application to flow computations on moving grids. AIAA J., **17**, 1030–1037
218. Thompson, M.C., Ferziger, J.H. (1989): A multigrid adaptive method for incompressible flows. J. Copmput. Phys., **82**, 94–121
219. Thompson, J.F, Warsi, Z.U.A., Mastin, C.W. (1985): Numerical grid generation - foundations and applications. Elsevier, New York
220. Tryggvason, G., Unverdi, S.O. (1990): Computations of 3-dimensional Rayleigh-Taylor instability. Phys. Fluids A, **2**, 656–659
221. Travin, A., Shur, M., Strelets, M., Spalart, P. (2000): Detached-eddy simulations past a circular cylinder. Flow Turbulence and Combustion, **63**, 293–313
222. Truesdell, C. (1977): A first course in rational continuum mechanics, vol. 1. Academic Press, London
223. Tu, J.Y., Fuchs, L. (1992): Overlapping grids and multigrid methods for three-dimensional unsteady flow calculation in IC engines. Int. J. Numer. Methods Fluids, **15**, 693–714
224. Ubbink, O. (1997): Numerical prediction of two fluid systems with sharp interfaces. PhD thesis, University of London
225. Vahl Davis, G., Mallinson, G.D. (1972): False diffusion in numerical fluid mechanics. Univ. New South Wales Sch. Mech. Ind. Engrg., Report 1972/FM/1
226. Van den Vorst, H.A., Sonneveld, P. (1990): CGSTAB, a more smoothly converging variant of CGS. Tech. Report 90–50, Delft University of Technology
227. Van den Vorst, H.A. (1992): BI-CGSTAB: a fast and smoothly converging variant of BI-CG for the solution of non-symmetric linear systems. SIAM J. Sci. Stat. Comput., **13**, 631–644
228. Van der Wijngaart, R.J.F. (1990): Composite grid techniques and adaptive mesh refinement in computational fluid dynamics. Report CLaSSiC–90–07, Dept. Computer Science, Stanford Univ.
229. Van Doormal, J.P., Raithby, G.D. (1984): Enhancements of the SIMPLE method for predicting incompressible fluid flows. Numer. Heat Transfer, **7**, 147–163
230. Van Doormal, J.P., Raithby, G.D., McDonald, B.H. (1987): The segregated approach to predicting viscous compressible fluid flows. ASME J. Turbomachinery, **109**, 268–277
231. Vanka, S.P., Leaf, G.K. (1983): Fully coupled solution of pressure linked fluid flow equations. Rept. ANL–83–73, Argonne Natl. Lab.

232. Vanka, S.P. (1983): Fully coupled calculation of fluid flows with limited use of computer storage. Rept. ANL–83–87, Argonne Natl. Lab.
233. Vanka, S.P. (1986): Block-implicit multigrid solution of Navier-Stokes equations in primitive variables. J. Copmput. Phys., **65**, 138–158
234. Wesseling, P. (1990): Multigrid methods in computational fluid dynamics. ZAMM - Z. Angew. Math. Mech., **70**, T337–T347
235. Weiss, J., Maruszewski, J.P., Smith, W.A. (1999): Implicit solution of preconditioned Navier-Stokes equations using algebraic multigrid. AIAA J., **37**, 29–36
236. White, F.M. (1986): Fluid mechanics. McGraw Hill, New York
237. Wilcox, D.C. (1998): Turbulence modelling for CFD. DCW Industries, Inc., La Cañada, California
238. Williams, F.A. (1985): Combustion theory: the fundamental theory of chemically reacting flow systems. Benjamin-Cummings Pub. Co., Menlo Park, CA
239. Yakhot, V., Orszag, S.A. (1986): Renormalization group analysis of turbulence. I. Basic theory. J. Sci. Comput., **1**, 1–51
240. Yoo, S.-D. (1998): Numerische Berechnung von Strömungen mit freien Oberflächen auf randangepaßten beweglichen Gittern, Dissertation, University of Hamburg, Germany
241. Zang, Y., Street, R.L., Koseff, J.R. (1993): A dynamic mixed subgrid-scale model and its application to turbulent recirculating flows. Phys. Fluids A, **5**, 3186–3196
242. Zang, Y., Street, R.L. (1995): A composite multigrid method for calculating unsteady incompressible flows in geometrically complex domains. Int. J. Numer. Methods Fluids, **20**, 341–361
243. Zhang, H., Zheng, L.L., Prasad, V., Hou, T.Y. (1998): A curvilinear level set formulation for highly deformable free surface problems with application to solidification. Numer. Heat Transfer, **34**, 1–20
244. Zienkiewicz, O.C. (1977): The finite element method, McGraw-Hill, New York

Index